UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE, INC. Defendant.	Case No. 3:10-cv-03561-WHA

OPENING EXPERT REPORT OF DR. OWEN ASTRACHAN ON TECHNICAL ISSUES RELATING TO FAIR USE

CONTAINS HIGHLY CONFIDENTIAL – ATTORNEYS' EYES ONLY MATERIAL SUBJECT TO PROTECTIVE ORDER

TABLE OF CONTENTS

I. INTRODUCTION

A. Case Background

1. Google has engaged me to provide expert background to assist the jury in understanding computer programming, application programming interfaces (APIs), and—in particular—the Java programming language and its APIs. Google has also asked me to address technical issues related to its fair use defense, including:

a. the purpose and character of the use;

b. the nature of the copyrighted work;

c. the amount and substantiality of the portion taken; and

d. the effect of the use upon the potential market.

2. I am being compensated for my work in this litigation at the rate of $500 an hour. My compensation does not depend in any way on the outcome of this litigation.

3. At this time, I have not created any exhibits to be used as a summary of, or as support for, my opinions. I reserve the right to create any additional summaries, tutorials, demonstrations, charts, drawings, tables, and/or animations that may be appropriate to supplement and demonstrate my opinions if I am asked to testify at trial.

4. I understand that discovery is ongoing in this case. I therefore reserve the right to supplement my opinions after I have had the opportunity to review deposition testimony or in light of additional documents that may be brought to my attention. Further, if Oracle or its experts change their opinions (either explicitly or implicitly) in such a manner as to affect my conclusions, I may supplement my opinions with any necessary rebuttal reports. Specifically, I understand that the Court has adopted an expert-discovery schedule that allows experts to file rebuttal and/or reply reports, depending on the burdens of proof.

B. Professional Qualifications

5. I am Professor of the Practice of Computer Science and Director of Undergraduate Studies in the Computer Science Department at Duke University. I earned my AB degree with distinction in Mathematics from Dartmouth College and MAT (Math), MS, and PhD (Computer Science) degrees from Duke University.

6. I teach undergraduate computer science courses using the Java, C++, and Python programming languages, and helped develop broadly-used teaching materials, including a C++ textbook and Java language programming exercises and documentation.

7. I received a National Science Foundation CAREER award in 1997 to incorporate design patterns in undergraduate computer science curricula, and an IBM Faculty Award in 2004 to support componentization in both software and curricula. That award helped to construct reusable assignments and software tools to support students across our courses at Duke. I was one of two inaugural NSF CISE Distinguished Education Fellows in 2007 to lead efforts to revitalize computer science education using case- and problem-based learning. That project was the precursor to an eight year NSF-College Board project on which I am the principal investigator to create a new Advanced Placement course in computer science that will launch in 2016.

8. My research interests have been built on understanding how best to teach and learn about object-oriented programming, software design, and computer science in general; and I am now working on developing a portfolio of substantial, interdisciplinary problems that help explain how computer science is relevant to students in the social and natural sciences.

9. My qualifications and information regarding my prior testimony are attached hereto as Exhibit A.

C. Documents and Information Considered

10. My opinions are based on my relevant knowledge and experience, the trial testimony and exhibits from the first phase of the original trial (which I attended daily), and the materials cited and discussed in the report. Additional documentation and information considered and relied upon is identified in Exhibit B.

11. I also have specialized knowledge of the matters set forth herein, and if called as a witness I could and would testify competently thereto.

12. My research and analysis of the materials, documents, allegations, and other facts in this case are ongoing, and if additional information becomes available through discovery and depositions in this action, I reserve the right to provide additional opinions.

II. BRIEF SUMMARY OF MY OPINIONS

13. "Java" may refer to various different things. These include, among other things, the Java programming language, the Java application programming interfaces (APIs), or software source code written in the Java programming language that references and implements the APIs. In this case, except for a very small number of files, Oracle did not allege infringement of the software source code referencing the APIs. Nor did it allege infringement because portions of Android or some applications for Android are written in the Java programming language. Rather, Oracle's infringement claim was based on Google's independent creation of software source code written in the Java language that references and implements Java APIs, and in doing so uses the "method declarations" of those APIs (described in detail below).

14. As explained in more detail below, an API provides programmers with a way to access the functionality of a software service. For example, most programming languages provide a way to calculate the square root of a number. Java does this by way of an API. In

order, for example, to calculate the square root of 25.0, a Java programmer includes the text math.sqrt(25.0) in the text of their program. The text "math.sqrt(a)"—where "a" is replaced by the variable or number that the programmer for which the programmer wants to calculate the square root—is the API for the square root "method."

15. APIs are implemented by software. For example, the sqrt(a) API can be implemented in many different ways. First, there are many different mathematical algorithms for calculating square roots (much as there are many different ways to make an apple pie). Second, there are many different ways to write the programming code that implements any given algorithm (just like two people who write down the same recipe can describe the various steps using different words and sentences). However, if one sets out to implement an API, the one part that cannot change is the API itself. For example, if one wants to implement the sqrt(a) API, one cannot change the method name from "sqrt" to, say, "squareroot." As I will discuss below, invoking an API using the Java programming language requires the use of the API's method declarations (names, data, and data types). To *implement* the corresponding API, one must include all of these exactly as the API requires them to be used. If even a minor change is made to the API, code that invokes the API will fail to operate. That said, there are potentially different ways to write the underlying implementing code for a given API.

16. It is my opinion that the facts of this case show that the first statutory fair use factor —the purpose and character of the use—weighs in favor of a finding of fair use. In brief, and as explained in more detail below, Android's use of the method declarations for the 37 Java API packages was a transformative use because it involved the provision of important and substantial changes to the way in which the method declarations had previously been used. Moreover, the method declarations were put to use in a manner that allowed other persons and

entities to further transform their use in new and different ways. Even Oracle itself recognizes the transformative nature of Android, and is now porting OpenJDK to Android.

17. It is my opinion that the nature of the copyrighted work demonstrates that the second statutory fair use factor also weighs in favor of a finding of fair use. As explained in more detail below, the allegedly infringed copyrighted work is a highly functional work that is used to carry out the execution of programs written in the Java programming language. Sun itself recognized that the Java programming language, to which the copyrighted work relates, was free and open for all to use and in fact encouraged its use in academic and other settings. Oracle understands that the portions of the copyrighted work that are allegedly infringed are fundamental to the language. And the allegedly infringed portions of the copyrighted work display only minimal creativity and are for the most part dictated by functional considerations, past industry practice, the common sense of programmers, and practical necessity. That is, it is my opinion that Google's use of these APIs is necessary to meet industry expectations and conventions associated with the Java programming language, and therefore also for programming efficiency and interoperability.

18. Relatedly, it is my opinion that any similarity between the names of elements (such as package, class and method names) in the method declarations of these APIs in the Java and Android platforms reflects functional considerations, and that any similarity between the organization of elements in the implementations of these APIs in the Java and Android platforms also reflects functional considerations. Indeed, many of the names of elements of the Java API were drawn from usage in other languages or platforms that pre-date Java. I discuss several examples of such borrowing of names by Java from other languages in section VI.B.5, below.

19. It is my opinion that the third statutory fair use factor also weighs in favor of a finding of fair use. The amount of the copyrighted work allegedly taken is very small—the method declarations form but a tiny part of the copyrighted Java SE platform, which also contains things like Sun/Oracle's virtual machine. And even if all of the additional copyrighted material is ignored in favor of an analysis of the 37 API packages, Google once again took the smallest possible portion of the material, using only the method declarations of the 37 API packages, while writing its own implementing code that often differs substantially from the implementing code of the copyrighted work.

20. Finally, it is my opinion that the fourth statutory fair use factor weighs in favor of a finding of fair use. Android could not have a substantial impact on the market for Java SE because Android was designed for mobile devices, a class of devices for which Java SE was and remains ill-suited. Moreover, Sun/Oracle itself had already decided to release Java SE under an open-source license (as a project called OpenJDK), which allows use of the 37 API packages without any requirement of payment or compensation of any kind to Sun/Oracle.

III. LEGAL STANDARDS

21. I understand that fair use is an affirmative defense against copyright infringement, meaning that if the use of copyrighted materials qualifies as fair use then there is no infringement.

22. I further understand that Section 107 of the Copyright Act lists four factors the courts "shall" consider in determining whether use of another's copyrighted work is a "fair use":

a. the purpose and character of the use;

b. the nature of the copyrighted work;

c. the amount and substantiality of the portion taken; and

d. the effect of the use upon the potential market.

23. I am informed that not all of the four factors need favor a use in order for it to be fair. I further understand that fair use is appropriate where custom or public policy at the time would have defined the use as reasonable, or where a reasonable copyright owner would have consented to the use.

24. I have also been informed that these statutory factors are not exhaustive. I further understand that the doctrine of fair use requires weighing the strength of each factor relative to other factors in light of the purposes of copyright.

25. For the first statutory fair-use factor—the purpose and character of the use—I understand that a key question is whether the allegedly infringing use is "transformative." That is, the issue is whether the allegedly infringing use "adds something new, with a further purpose or different character, altering the first with new expression, meaning, or message." *Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569, 579 (1994).

26. I understand that the second statutory fair use factor—the nature of the copyrighted work—calls for recognition that some works are closer to the core of intended copyright protection than others. Accordingly, to the extent a computer program is "essentially utilitarian" or "largely functional," it receives relatively weak protection. Along those lines, I have also been informed that computing instructions are generally considered functional, even if covered by copyright, if they are the only and essential means for accomplishing a given task.

27. I further understand that the third fair use factor—the amount and substantiality of the use—asks whether the defendant used more of the copyrighted work than it needed to in light of the purpose and character of the use. That is, if the user of the copyrighted work only copies as much as is necessary for their intended use, this factor will not weigh against them.

28. Finally, I understand that the fourth fair use factor is the effect upon the potential market or actual market for the copyrighted work. To that end, I understand that a transformative work is less likely to cause a substantially adverse impact on the potential market for the original. Along those lines, I understand that in the case of a transformative use, market substitution is less certain such that market harm cannot be presumed or inferred. I further understand that suppressing demand through means such as parody or criticism is not a market harm under the fourth factor, which is instead focused on the harm caused by actions that usurp the market for the original.

IV. TECHNICAL BACKGROUND

A. Programming Languages Generally

29. Programming languages allow programmers to express ideas in words and symbols that are ultimately executed as code. Programming languages can differ in grammar and syntax just as spoken languages differ. However, high-level concepts are typically very similar across programming languages. Programmers fluent in one language have an easier time learning a new programming language than do people learning a new natural language.

30. Consider the German word *Geländesprung* defined:

> **Geländesprung:** a jump made in skiing from a crouching position with the use of both poles.

An idea whose expression requires many English words can be expressed in a single German word. Different programming languages and different libraries illustrate similar differences in how algorithmic ideas are realized in programs. Just as *chat* means "to converse informally" in English and means "a small domesticated feline (cat)" in French,

so might *00010100111010* instruct one computer to add two numbers and another computer to print the letter *q*.

31. Rather than instruct computers at this level of zeros and ones, languages have been developed that allow ideas to be expressed at a higher level—in a way more easily understood by people. Just as translators can translate English into both Japanese and Swahili, so can translating computer programs like compilers and interpreters translate a high-level language into a low-level language—ultimately into zeros and ones—for a particular computer.

32. Consider a Java and a C++ program to perform the same task: find the unique words in a textfile. The two programs below print in alphabetical order each unique word in a text-file storing Melville's Bartelby the Scrivener. First the Java program, which is written in the Java programming language:

```
import java.util.*;
import java.io.*;
public class Unique {
    public static void main(String[] args) throws IOException{
        Scanner scan = new Scanner(new File("/data/melville.txt"));
        TreeSet<String> set = new TreeSet<String>();
        while (scan.hasNext()){
            String str = scan.next();
            set.add(str);
        }
        for(String s : set){
            System.out.println(s);
        }
    }
}
```

The equivalent C++ program, which is written in the C++ programming language, follows. This program generates exactly the same output as the Java program.

```
#include <iostream>
#include <fstream>
#include <set>
using namespace std;
```

```
int main(){
  ifstream input("/data/melville.txt");
  set<string> unique;
  string word;
  while (input >> word){
    unique.insert(word);
  }
  set<string>::iterator it = unique.begin();
  for(; it != unique.end(); it++){
    cout << *it << endl;
  }
  return 0;
}
```

33. The programs share many characteristics. A Java programmer who had never seen C++ would not be able to write the C++ program, but would likely be able to read it and understand the general ideas in it, and vice versa for a C++ programmer unfamiliar with Java.

34. Each program begins by including references to libraries (equivalently, packages—explained below) that are used in the programs. The Java program uses two packages: java.util and java.io; the java.lang package is included automatically. The C++ program references three included header-files, the C++ equivalent of a package, explicitly: iostream, fstream, and set (other libraries, like <string>, are included as part of these three, since in C++ one header file can include other header files).

35. Both programs use a while loop to read the text file and a set to store the unique words read. In Java the word "add" is used to store a word in the set, whereas the word "insert" is used in C++. Both languages use curly braces { } and semi-colons as syntactic delimiters in writing the programs. Both programs use a for loop to print the words stored in alphabetical order.

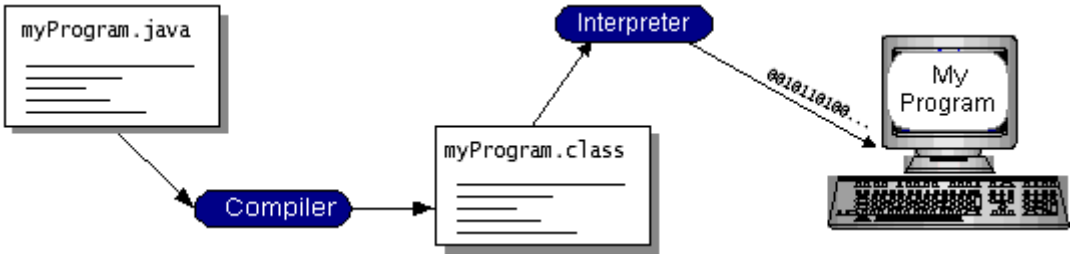
36. Although the programs share many characteristics, they are executed very differently. The Java program is translated into bytecodes, which are executed by a virtual

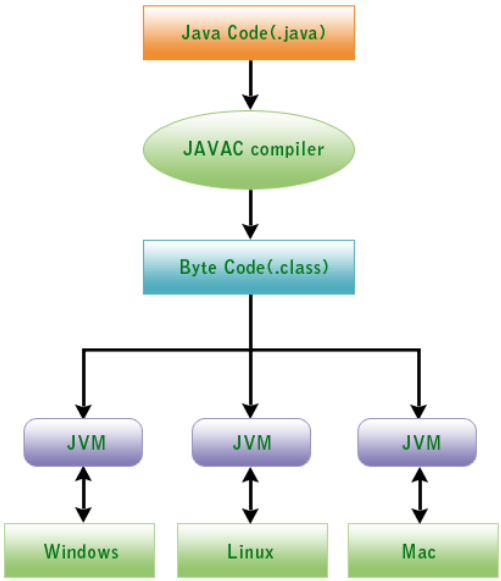
machine.[1] The C++ program is translated into assembly code specific to the computer on which it executes. The Java bytecode is 49 lines:

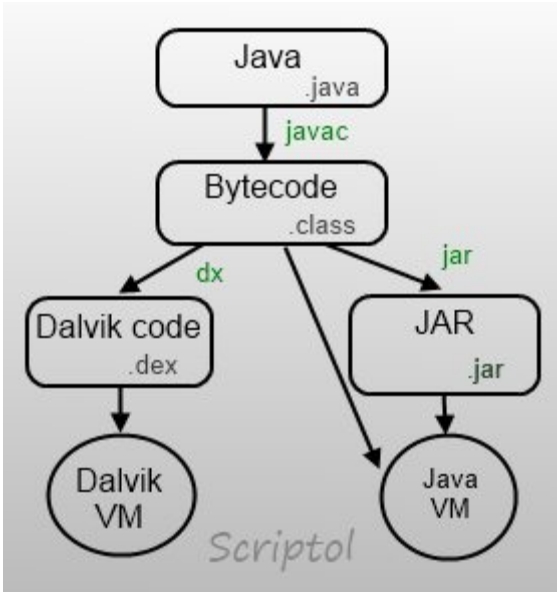
```
public class Unique {
  public Unique();
    Code:
       0: aload_0
       1: invokespecial #1
       4: return

  public static void main(java.lang.String[]) throws java.io.IOException;
    Code:
       0: new           #2
       3: dup
       4: new           #3
       7: dup
       8: ldc           #4
      10: invokespecial #5
      13: invokespecial #6
      16: astore_1
      17: new           #7
```

[1] To aid the jury, I may provide some background regarding the basic concepts of virtual machines, bytecode, and runtime interpretation of bytecodes by a virtual machine. The following diagrams illustrate the basic teachings:







```
      20: dup
      21: invokespecial #8
      24: astore_2
      25: aload_1
      26: invokevirtual #9
      29: ifeq          46
      32: aload_1
      33: invokevirtual #10
      36: astore_3
      37: aload_2
      38: aload_3
      39: invokevirtual #11
      42: pop
      43: goto          25
      46: aload_2
      47: invokevirtual #12
      50: astore_3
      51: aload_3
      52: invokeinterface #13,  1
      57: ifeq          82
      60: aload_3
      61: invokeinterface #14,  1
      66: checkcast     #15
      69: astore        4
      71: getstatic     #16
      74: aload         4
      76: invokevirtual #17
      79: goto          51
      82: return
}
```

37. In contrast, the C++ assembly program—the equivalent of the Java bytecode—is roughly 9,500 lines of code (though because it is not interpreted by a virtual machine, it generally executes faster).

38. There are hundreds of programming languages. Some of the most popular contemporary languages include C, C++, Objective C, Python, Perl, and Swift, as well as Go and DART. Precursor languages, some of which remain in use today, include FORTRAN, Pascal, and BASIC. Similar to other languages (*e.g.*, English, French, and Spanish) each programming language has its own unique rules and conventions, though there are overlaps between and among them.

39. Many languages are also associated with a corresponding set of core libraries. For that reason, in everyday usage most computer scientists would understand the "language" to include the corresponding set of core libraries.

40. When the programming language C was introduced in 1978, the first program was a simple three-line program that printed "hello world" on an output device such as a screen or printer. Since programs typically must generate output for the results to be visible, the so-called HelloWorld program became an iconic example of a simple program that typically used an input/output (I/O) library to execute. The Java version of HelloWorld is:

```
public class Hello {
    public static void main(String[] args){
     System.out.println("hello world");
    }
}
```

This simple program uses the class java.lang.String (for the string "hello world"), the class java.lang.System and its field java.lang.System.out, an instance of the class java.io.Printstream, for printing the string to the user's screen.

41. In a more modern introduction to programming the HelloWorld program might be expanded. In a recent online Java course offered through the Coursera platform, this example is expanded to print "hello world" in as many as 40 languages. This requires using additional APIs/packages, but is more international and offers more to students learning to program today.

```
import edu.duke.FileResource;
import edu.duke.URLResource;

public class HelloWorld {

     public void runHello(){
          FileResource hello = new FileResource("hello_unicode.txt");
          for(String line : hello.lines()){
               System.out.println(line);
          }
     }
```

```
    public static void main(String[] args){
        HelloWorld hw = new HelloWorld();
        hw.runHello();
    }
}
```

See https://www.coursera.org/learn/java-programming.

B. APIs Generally

1. What is an API?

42. An application programming interface (API) is a set of vocabulary and grammar that allows one program to talk to another program. The second program's "API" is used by the first program to ask the second program to do things for it. Because computers are very literal, an API must be followed scrupulously—even small deviations typically result in errors.

43. Various dictionary definitions of APIs capture the idea that an API provides a specified and documented mechanism to invoke, operate, and interact with software services.

a. "Software that an application program uses to request and carry out lower-level services performed by the computer's ... operating system." Newton's Telecom Dictionary, 25th Edition.

b. "The specification of how a programmer writing an application accesses the behavior and state of classes and objects." Sun Java Glossary (*available at* http://www.oracle.com/technetwork/java/glossary-135216.html; *previously available at* http://java.sun.com/docs/glossary.html).

c. "In most procedural languages, an API specifies a set of functions or routines that accomplishes a specific task or are allowed to interact with a specific software component." Wikipedia, *Application programming interface* (*available at* https://en.wikipedia.org/wiki/Application_programming_interface) (as of December 20, 2015); *see also* Wikipedia, *Application programming interface* (*available at* http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=437864024) ("a particular set of rules and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.") (as of July 13, 2011).

Although these definitions use different terminology, they are not materially different from, and are in fact consistent with, my summary above. The interface itself is implemented by software, *i.e.*, by source code that is written to provide the functionality of the interface.

44. APIs are an abstraction. That is, they are less complex than the code that implements the APIs. In fact, the APIs themselves are not executable code. They are shortcuts, or labels, that refer to the underlying software, sort of like a system of car parts in a catalog might have names or numbers that are then used by a mechanic to actually find and then use the underlying parts. When software is compiled, these labels are used to access the actual software that implements the functionality, but the labels (the APIs) do not implement the functionality themselves.

45. When used alone, the term API most precisely refers to the set of rules and specifications. That said, the term API is also sometimes used colloquially to refer to the software that implements the rules and specifications and therefore is operated by the communication from another program, as explained in more detail below. API can also refer to either a specific "element," "component," or functionality within the API, or to a collection of them. This report will generally use API to mean such a collection, and "API elements" or "API components" to refer to the individual mechanisms within the collection.

46. APIs are similar to the interfaces that a computer user uses to operate software, like a keyboard command or button. In each case, the person using the program does something to inform the program that they want a specific action to happen, and then that action happens. No deep expertise or understanding of the inner workings of the computer system is needed by the person seeking to use the program. For example, a computer user might type the "Ctrl+P" key combination or click an icon that looks like a printer, and then, in the dialog box that

appears, choose the file to be printed and the number of copies that should be printed. Typing "Ctrl+P" or clicking the icon would invoke the underlying printing functionality, and (once the number of copies is specified) cause the software to print the document that number of times.

47. The user does not need to have substantial understanding of the underlying printing mechanism, they just need to learn and remember the familiar "Ctrl+P," give the necessary information (*e.g.*, number of copies they want printed), and the computer takes care of the rest. For that reason, a user can continue to rely on the common API for printing—"Ctrl+P"—even if and when they connect a new printer. The user need not understand how the different drivers and printer software communicate between the computer and the printer; rather, they can rely on the known, customary command structure. In fact, they would likely find it rather inconvenient if instead of being able to continue using the "Ctrl+P" API, they had to instead learn to use a different—and possibly less intuitive—command.

48. Similarly, when invoking or using an API in a software application, a programmer should not need to review or understand the underlying implementation or source code for the API, as that code has already been written. Like using "Ctrl+P" to print, they only need to know the name and functionality of the API. In order to write software that prints, a programmer would read and learn their chosen operating system's API for printing, and then invoke that API from their program, telling the API critical information like what document to print and how many copies to print. Just like typing the "Ctrl+P" command or clicking the printer icon, using the name of the API element in the software invokes the underlying functionality of the API and causes printing to happen, without the programmer needing to have a deep knowledge of the particular mechanisms that allow the API to function. Among other benefits, this means that a programmer can use an API to create software that works on different printers (color, black and

white, inkjet, etc.) without knowing in detail how those different printers work, as long as the underlying implementation supports them.

49. An API also can be analogized to the interface for driving a typical car. Every car has a variety of elements that are part of the overall system of communication and operation that a driver must understand and use in order to drive the car. These elements include the gear lever, the turn-signal stalk, the steering wheel, and the accelerator and brake pedals. Some of these elements provide information from the driver to the car, while others provide information from the car to the driver, and others do both things at the same time. In combination, these elements form the interface to the "application" that is the car, allowing a driver to "program" the car to do their bidding by using those elements to operate the car.

50. Thus, for example, the driver of a car can make it accelerate by pressing the accelerator. The further the driver presses the accelerator, the faster the car speeds up. The accelerator can be thought of as an API for the car that makes the car go. Every car that implements this API will have an accelerator, and each of them will share in common the fact that the car speeds up faster the further the accelerator is pressed down.

51. So long as a driver understands this functionality—that the rate of speeding up a car depends on how far the accelerator is pressed down—the driver need not know *how* this happens. Similarly, if a driver understands how the other interfaces to the car's operation function (the turn signals, steering wheel, etc.), the driver need not know how the engine, light bulbs, or transmission work.

2. What is the purpose of an API?

52. The primary purpose of APIs is to allow one piece of software to speak to another piece of software in a clearly defined, reusable, interoperable way. This simple goal has a number of important ramifications and benefits.

53. Familiar interfaces make it simpler to use things and to use them more proficiently. When using a new car, most drivers do not think about how that particular steering mechanism works. For example, the wheels of the car might be turned by a rack and pinion system or a recirculating ball system. But the steering wheel itself functions the same way regardless of how the internal steering mechanism and system is designed, *i.e.*, when the driver turns the wheel to the left, the car turns in the left direction. This interface—the same, familiar steering wheel—facilitates using the car, regardless of which specific car is being used.

54. The same thing happens in software—using (or providing) a standard API allows the users of that API (software programmers) to move between any software platforms that provide the same API, because their familiarity and existing skills in using that particular interface transition over.

55. A defined, fixed API allows for different underlying implementations, which gives users the ability to move from one implementation to another. In this way, APIs promote innovation and choice in the software market. For example, if a software platform provides a set of APIs, a subsequently created platform that implements those same APIs (for example, by writing different source code to implement those APIs) can transform those APIs by using them in a different context. Developers who use the platform would therefore already be familiar with the APIs, and so could leverage their existing knowledge. In our printing analogy, another program that provides the same commands (like "Ctrl+P") will be familiar and therefore easier to use. In fact, once a user is accustomed to the "Ctrl+P" convention, they may be confused if a program uses something else to control printing, or "Ctrl+P" invokes a different function (*e.g.*, the common Paste function).

56. APIs also help insulate software programmers from underlying complexity. This is referred to as "encapsulation." In the car example, the internal steering system can be changed specifically because the familiar steering wheel interface (the car's "API") has hidden these implementation details from the driver. This shielding and simplification is an important part of what an interface provides—most users do not need to know the details; it has allowed car manufacturers to switch from old technologies to new ones without introducing a new learning curve for consumers. The concept that moving the wheel counter-clockwise turns the car left remains the same, and that allows consumers to rely on this familiar concept, regardless of which car they use.

57. APIs also help programmers and the industry by allowing software to be reused. This is important; new code is difficult and expensive to write and test, and so individual programmers and corporations like to reuse code as much as possible. Even if they cannot reuse an entire program (say, because the two pieces of hardware are very different, as they are between a desktop computer and a phone) they still prefer to reuse as many parts of the software as possible. APIs help make this possible by allowing the same basic functionalities to be provided and used in a replicable, but portable, way.

58. The API for a programming platform exists to extend a programming language. The goal is to allow programmers to express themselves using new words. This is similar to developing new words for languages that human beings use; for example, the word "selfie" was recently adopted as the Oxford Dictionaries Word of the Year 2013. *See* http://blog.oxforddictionaries.com/press-releases/oxford-dictionaries-word-of-the-year-2013/. In short, the goal of a programming language is to empower others to write computer programs

more quickly and easily by making sure the programmers have easy access to the right words and ideas that they need.

3. How do programmers use APIs?

59. An API typically consists of what programmers call method declarations or, equivalently, function declarations.[2] (I discuss the declaration components in further detail in the next section.) The declarations define the means by which a programmer invokes the functionality provided by what is known as a "method" or, equivalently, a "function." These two terms are synonyms, used somewhat interchangeably depending on the programming language. Java programmers (and therefore this report) use the term "methods"; in contrast, C programmers typically use the term "functions." Both represent the same thing—a piece of software that performs a specific task and can be invoked when needed. Method declarations are the primary mechanism by which programmers invoke the functionalities provided by a software system.

60. In the user interface analogy, "Ctrl+P" and the printer icon are analogous to a declaration—they are both mechanisms to invoke the print function. A user interacting with a software program might use a menu or a menu shortcut to open a file, or to save or print what has been opened, *e.g.*, in a word processing program. Often an icon of a printer or a disk can be pressed to invoke the same function as choosing a menu item or the menu shortcut. In all three cases—pressing the icon, choosing the menu item, or typing a keyboard shortcut—the same underlying software is invoked and causes a specific action—printing the file or saving it, for example. The method declarations in an API are directly analogous—just as the user might click on an icon or press a sequence of keys to use a keyboard shortcut to invoke more complicated

[2] The Java Language Specification uses the term "method header" instead of "method declaration"; however, consistent with both parties' usage in the first trial, I use the term "method declaration" in my report.

operations such as printing or saving, the program calls the function or method to invoke and control a more complicated service or feature provided by the underlying software.

61. For example, when a Java programmer wants to get the square root of 25, their program will have to contain the following text:

```
Math.sqrt(25.0)
```

This will cause the underlying system to do the math and tell the program that the answer is "5.0".

62. Similarly, to get the absolute value of -25, the program must contain

```
Math.abs(-25).
```

This will cause the underlying system to do the math and tell the program that the answer is "25."

63. As another example, when a Java programmer wants to calculate the larger of, for example, the two numbers 2 and 4, their program will have to contain the following text:

```
Math.max(2, 4)
```

This will cause the underlying system to compare the values and tell the program that the answer is "4."

4. What are the components of a method declaration?

64. Every method has several important characteristics that collectively are referred to as the "method declaration." The first is simply the method's *name*. Method names describe the purpose of the method, so that a programmer can easily memorize and recognize the purpose from the method's name, and vice-versa. A simple example of this is the method above named "abs," so named because its function is to calculate the absolute value of a number. To use a method, the programmer must know the method's name. If the programmer does not know the

precise name, or knows only something similar, they cannot use the method, because the software cannot guess at what the programmer meant. For example, if a Java programmer writes "squareroot(25.0)" instead of "sqrt(25.0)," this will result in an error instead of calculating the square root of 25.0.

65. The second important characteristic of essentially every method declaration is the set of *arguments* that the method expects to receive when invoked. When the method is called, the programmer typically provides information to the method that informs the software exactly what the programmer wants to happen, just as a user must usually specify how many copies they want printed after they click the print button. The information provided to the method is called an argument (or parameter), and a method is said to "accept" the permitted arguments. The ability of a method to accept an argument as an input is what allows a general purpose method to act on specific data.

66. For example, think of the "plus" or "add" button on a calculator. This is a "general purpose" button—it can add any numbers one can type in, not just one specific set of numbers. If one thinks of the "plus" button on a calculator as a method, the numbers one asks the calculator to add (say, 2 and 2) are the arguments to the "plus" button—those arguments determine the specific outcome of the general purpose button. Similarly, the number of copies one tells the print dialog (or print method) to print is also an argument—it again tells the general purpose function ("print") a specific behavior ("print two copies."). In the "abs" function mentioned previously, there is only one argument, and that argument is simply a number, whose absolute value the program wishes to calculate. A slightly more complicated example is the

"max" function, also discussed above. "max" takes in two arguments (2 and 4 in the above example), so as to determine which is numerically larger and return that value.[3]

67. These arguments or parameters must be defined when the API is first created, and are typically limited. For example, it would not make sense to ask one's printer to print "hippopotamus" number of copies of a document—that argument must be a number. In fact, the definition of a method in many languages, including Java, will indicate what "type" of argument a function will accept, such as an integer, a string, or another data type. A steering wheel, similarly, can accept arguments such as clockwise or counter-clockwise, as well as how much to turn (*e.g.*, in degrees), but cannot accept inputs such as "up," "down," or "hippopotamus." The functionality of each method constrains what parameter(s) are acceptable, and if the proper parameters are not passed to the method, any attempt to use the method will fail.

68. When a program invokes a call to a method and passes it the arguments, the method then typically returns a *result* that the program can use for other purposes. This result is the final important characteristic of the method declaration, and is called the *return* (sometimes the *return value).* In the calculator example, where plus is the method and the arguments are 2 and 2, the return value will be 4 (that is, 2+2 returns 4). For the abs method, which computes the absolute value of a number, when the argument is 2 or -2, the return value will be 2. And for the max method when computing which is larger, 2 or 4, the return (or, in other words, the result of applying the method) will be 4.

69. The purpose of the return value is to return information that can be used by the program for other purposes. For example, after telling a calculator to add 2 and 2, the calculator returns "4," which one can use as the first step in the next math problem one intends to solve.

[3] There are actually four variants of max: int, long, double, float. They each take two arguments and return the larger. Variants are only for types of arguments.

Similarly, the return may be a message indicating the status of a method—for example, a "print" method might return "OK" (telling the program that the printing functionality has successfully completed) or "OUT OF PAPER" (telling the program that the printing functionality has hit a snag). These status messages would in turn be handled by other methods, possibly doing something like popping up an error message, or silently concluding that all is well and allowing the user to continue with their work.

70. To summarize, each method declaration has three parts—the name, arguments, and return. As a shorthand, programmers may refer only to the name of the method, but to fully know what method they are describing, it is necessary to know all three parts of the method declaration. They can be defined succinctly as:

name:	the method name, which indicates its purpose and is used by a programmer or program to call or invoke the method.
argument:	the data on which the method acts. The data passed as an argument to a method is often manipulated and referred to within the method itself.
return:	the result of calling the method with specific arguments. This is "returned" to the programmer.

71. The documentation for a method will combine these pieces to form a reference for programmers using the API. For example, the brief version of the documentation for the abs method is:

int abs(int a) Returns the absolute value of an int value.

While this may not be easy for a non-programmer to understand, it is quite straightforward to a programmer:

- The first part ("int") shows that the return will be an "int" (short for an "integer"; *i.e.*, a number). This tells the programmer what type of result to expect when using the function.
- The second part ("abs") is the name of the method.

- The part in the parentheses ("int a") is the argument. Again, this uses "int" to indicate that a single integer is expected; if the abs method is given something other than an integer, such as "hippopotamus," an error will occur. (The letter "a" is a convenient name for the argument, and can be changed without affecting compatibility.)

- The first, second and third parts discussed above comprise the "method declaration." The final part is a brief explanation of what the method does.

In combination, this short statement will allow a programmer to know how to use "abs" in their program to find the absolute value of a number.

5. How are APIs organized?

72. Related methods are often grouped together to make them easier to use, frequently into groups called (depending on the programming language) "libraries" or "packages." In Java, these groupings can be packages (the highest-level grouping, typically containing many classes), sub-packages, or classes (the lowest level of grouping, typically containing a handful of related methods). (*See, e.g.* "The Java Platform: A White Paper," Douglas Kramer, May 1996, *formerly available at* http://java.sun.com/docs/white/platform/javaplatform.doc1.html, and "Package Members" in The Java Language Specification, Third Edition, *available at* http://java.sun.com/docs/books/jls/third_edition/html/packages.html#7.1, for discussion of packages, class, and methods.)

73. To put it a different way, the API packages include subparts or files known as classes, and within these classes are methods. As an analogy, one can think of menus (like "File," "Edit," etc.) as "packages" of menu items, which organize the menu items so that they are grouped together in reasonable groupings. For example, to print using the menu, a user needs to know that Print is under File, rather than under Edit. As another analogy, one might consider a class library to constitute a multi-drawer file cabinet, within which a drawer might represent a

package, the hanging files represent different classes, and separate folders within each hanging file might represent different methods.

74. As an example of this grouping, take the "sqrt" method, which calculates the square root of a number. This method is typically grouped together in a library or package with methods that implement other, related mathematical functions, such as "sin" and "cos" (short for the trigonometric functions sine and cosine). "sqrt" has been grouped with other math functions since at least the ALGOL programming language in 1968, and is still grouped together with them in Java and other modern languages, such as the Python and Ruby languages. In fact, in Java, Ruby, and Python the class or module name is the same: Math (or math).

75. Specifically, the java.lang package (according to Oracle's documentation) "[p]rovides classes that are fundamental to the design of the Java programming language." *See* http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/package-summary.html; *see also* The Java Application Programming Interface, Volume 1: Core Packages (describing the core packages, including java.lang, java.io, and java.net as "the general-purpose libraries fundamental to every Java program"). One of these "fundamental" classes is the "Math" class, which Oracle describes as containing "methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions." The actual methods contained with the class are listed in this chart:

Method Name	**Functionality of the Method**
abs	Returns the absolute value of the argument. (Four variants)
acos	Returns the arc cosine of the argument.
asin	Returns the arc sine of the argument.
atan	Returns the arc tangent of the argument.

Method Name	Functionality of the Method
atan2	Converts rectangular coordinates (two arguments) to polar coordinates.
ceil	Returns the smallest integer that is not less than the argument, *e.g.*, if the argument is 1.9, will return 2.
cos	Returns the cosine of the argument.
exp	Returns *e* raised to the power of the argument.
floor	Returns the largest integer that is not more than the argument, *e.g.*, if the argument is 1.9, will return 1.
IEEEremainder	Returns the remainder of two arguments as prescribed by the IEEE 754 standard.
log	Returns the natural logarithm of the argument.
max	Returns the greater of two arguments, *e.g.*, if the arguments are 3 and 4, will return 4. (four variants based on types of arguments)
min	Returns the lesser of two arguments, *e.g.*, if the arguments are 2 and 3, will return 2. (four variants)
pow	Returns the value of the first argument raised to the power of the second argument.
random	Returns a random number between 0 and 1.
rint	Returns the closest integer to the argument.
round	Returns the closest number to the argument.
sin	Returns the sine of the argument.
sqrt	Returns the square root of the argument.
tan	Returns the tangent of the argument.
toDegrees	Returns the result of a conversion of the argument (an angle in radians) to degrees.
toRadians	Returns the result of a conversion of the argument (an angle in degrees) to radians.

76. Conversely, methods that are unrelated are not typically grouped together—for example, a method that prints text to a screen would not typically be in the same class or package as "sqrt."

77. To use a particular method, the Java programmer has to know what class, and what package, the method is in. A programmer calling the square root method in Java, for example, needs to know that the method is in the class named "Math," and the method is named "sqrt." This is expressed in the Java language by combining the two names as "Math.sqrt." In Ruby the same expression is used: "Math.sqrt"; in Python, the expression is "math.sqrt." The programmer also must know the other parts of the declaration—specifically that the method takes one argument (a number) and returns the square root of the argument. Once the programmer knows these things, the underlying functionality can easily be invoked, allowing the programmer to focus on the more complex task of writing their own software.

78. In Java, "public" methods are part of a class's API. These public methods can be called and used by programmers in writing programs that use a class via its API, *e.g.*, as in calling Math.sqrt. In some cases, the code implementing a public method may be a few lines. In other cases, a public method's implementation might be hundreds of lines of code. Sometimes these hundreds of lines may be placed into methods shared by other public methods in the same class. Re-using code between methods is an essential aspect of ensuring reliability and correctness in programs. Such shared methods are not part of a class's API, but instead are part of the implementation and are therefore not available for programmers to call and use in writing a program. Java uses the modifier "private" in method headers to indicate that these methods cannot be called by programmers writing programs using a class's API, but rather may only be used in implementing the public API. Thus, these methods can be called from within the class,

as part of the implementation of the class, but cannot be called by programmers referencing the class.

6. What is the distinction between an API and its implementation?

79. For every API, including the Java APIs at issue in this case, there is the API itself and an implementation for that API. In the Java programming language, APIs consist of method declarations (aka "headers"), comprising the elements mentioned above—name, arguments, and return; the implementation of the API includes those three elements as well as the program logic that actually performs the steps necessary to accomplish the purpose of the method.

80. Independent of, but related to, a given API's method declaration is an implementation of that API—the actual underlying source code that performs the functionality invoked by the API. An API can be implemented in more than one way, *i.e.*, the underlying program logic for the functionality of the API can differ. But each implementation must necessarily have the same method declaration elements—the package names and related method elements—in order to work properly. If these elements are not present and identical in different implementations of the same API, programmers will not be able to use the same names and structures when using the API. If the API is changed—that is, if the method declaration is changed—software that references the original API will not work at all or as intended.

81. To see why using the same names and structures is important, it may be useful to analogize this to non-programming languages. It is only by having a common vocabulary of words like "truck" that people can speak to each other. If the language is changed, even slightly—as it is when a speaker of American English uses "truck" while a speaker of British English uses "lorry"—then confusion arises. For programmers, similar confusion would occur if two different implementations of the same API used different names, arguments, and return values. For software, the result would be even worse than mere confusion, since computers

cannot guess at what the original software meant to say. Instead, faced with a similar situation, software might fail to execute altogether. Different implementations of the same API must use these same declaration elements, otherwise they are not implementing the same API and software that uses the API would not work with both implementations.

82. In contrast, the source code for the different implementations can differ. For example, different programming languages can be used to implement a particular API. Furthermore, any two given implementations, other than the parts required for compatibility (*i.e.*, the elements of the method header), will often be different if written by different programmers. However, because the implementations are constrained by the API as well as practical considerations such as programming efficiency and the underlying hardware, they will have similarities. For example, since cars are constrained by the requirements to have a steering wheel, gas and brake pedals, and four tires, they will often appear similar, featuring an engine, drive train, and brakes. But an expert will be able to distinguish a V8 from a V6 or direct fuel injection from a carburetor. Similarly, the source code implementing an API may appear similar to another implementation of the same API because of practical programming constraints. At the same time, implementations can also be quite different—the digital equivalent of choosing between a fuel-efficient but slow four-cylinder (or even a hybrid) versus a hungry but powerful V8.

83. To take one example, consider the "abs" method in the "Math" class in the java.lang package. The Android source code that implements the "abs" method documented above is:[4]

```
package java.lang;
```

[4] Android code cited in this report is publicly available at http://source.android.com. Unless otherwise noted, I use examples from a prior version of Android known as "Gingerbread" (also known as Android version 2.3), which was the exemplary version I used in my prior reports; however, the same principles apply to later versions of Android.

```
...
/**
 * Class Math provides basic math constants and operations such as
trigonometric
 * functions, hyperbolic functions, exponential, logarithms, etc.
 */
public final class Math {
        ...
    /**
            * Returns the absolute value of the argument.
            * <p>
            * If the argument is {@code Integer.MIN_VALUE}, {@code
Integer.MIN_VALUE}
            * is returned.
            *
            * @param i
            *            the value whose absolute value has to be
computed.
            * @return the argument if it is positive, otherwise the
negation of the
            *         argument.
            */

    public static int abs(int i) {
        return i >= 0 ? i : -i;
    }
```

The first line, "package java.lang;" is the name of the package of API elements in which the abs method resides, and indicates that this file contains a class which is part of that package. "public final class Math" specifies that the class named "Math" contains the method named "abs."[5] Both of these lines (which appear above in black), in this exact form, must be present in order to accurately implement the API, so all implementations of the Math.abs function will contain these two lines.

[5] "public" means that the Math class can be used by code outside of the java.lang package. "final" means that the "Math" class cannot be "sub-classed."

84. The lines of text that begin with asterisks (which appear above in blue) are programmer *comments*. Comments do not affect the compiled code that is distributed to users; instead, they explain to other programmers what the code does. In this case, the comments describe to a programmer the functionality of this method, and may also contain information about how to use the method.

85. Finally, the actual source code for the method—that is, the code that is ultimately compiled—is shown here in green and red. It starts by repeating the method declaration—"public static int abs(int i)" (in green)—also sometimes called the "header" or "signature." It then presents the program logic for the method—the single line "return i >= 0 ? i : -i;" (in red). (The variable name, here "i," is not part of the definition, and so can be different between different implementations without impacting compatibility.) This red portion, when compiled, is what actually tells the computer how to perform the method's functionality. In this case, the program logic could be stated in English as "if the given number is greater than or equal to 0, return that number; otherwise, return that number but with the opposite sign." Because creating the absolute value is simple, this program logic is brief; many more lines of program logic may be needed for more complicated methods.

86. Of this substantial amount of text, only the single line "public static int abs(int i)" (the method's signature, underlined above) is identical between this implementation of abs (in Android) and Oracle's implementation of abs (in the works at issue).

87. In fact, an essentially identical declaration is present in any implementation of the abs method in java.lang.Math. For example, Oracle has released an open source implementation of Java SE (typically referred to as OpenJDK); the non-profit GNU Project has written a Java implementation called GNU Classpath; and the non-profit Apache Foundation has written an

implementation of Java SE called Apache Harmony. *See, e.g.*, http://openjdk.java.net/ (OpenJDK homepage) and http://openjdk.java.net/faq/ (explaining open source license nature of license); GNU Classpath documents at http://www.gnu.org/software/classpath/docs/cp-hacking.html and Apache Harmony documents at http://harmony.apache.org/faq.html and http://harmony.apache.org/subcomponents/classlibrary/compat.html.

88. For standardization and compatibility reasons, the "abs" function discussed earlier has the following identical method declaration not only in Java and Android, but also in Harmony and GNU Classpath:

Java:	public static int abs(int a)
Harmony:	public static int abs(int i)
GNU Classpath:	public static int abs(int i)
Android:	public static int abs(int i)

The similarities are not limited to the abs method. Each of these projects implements the API packages at issue in this case, using the same package, class, and method names. Moreover, Oracle presumably named this method "abs" in part to increase efficiency and ease of learning for programmers who were familiar with other preexisting programming languages, since this name has been used for this function in many older programming languages, such as C.

7. Basic Example of a Java Method Usage

89. When a programmer is writing an application, and wants to use a particular functionality, they must invoke the functionality by using the appropriate method. If a programmer writing in the Java programming language wants to use Java's square root functionality to find the square root of 25, they would do that by incorporating the following language in their program:

```
double result = Math.sqrt(25.0);
```

90. The *argument* 25.0 is passed to the method Math.sqrt when the method is called, and "5.0" is *returned* by the method. In this example, the return value is then stored in the variable (of the "double" type) named "result" for use elsewhere in the program.

91. To write this example, a programmer who had never used Java would expect a square root method to be grouped with other mathematical methods. In many programming languages, including Java, these methods (or functions) are organized into a collection (*e.g.*, a library or class) named "Math." Finding the familiar "sqrt" method in the java.lang.Math class, this programmer might then read the corresponding documentation for that method to understand what result is returned and what special cases need to be considered. And, once familiar with this API, they may no longer have to refer to the documentation.

92. Note that at no point does the programmer need to know how the program logic that is invoked by the sqrt method actually calculates the square root—it could use a Newton-Raphson method, logarithms, or another mathematical algorithm for calculating the square root. As previously noted, these details are "encapsulated"—hidden behind the scenes. This focus on knowing and understanding the API, rather than understanding how the API's underlying program logic works behind the scenes, allows programmers to work more efficiently.

C. The Java Platform

93. Java's specifications were published or made publicly available in various forms, including in books and on the Java website, starting with the release of version 1.0 in 1996. Several revisions have been released since then. Note that, at times, Java version 1.5 has also been referred to as Java version 5.0. For consistency, I will generally refer to it as Java 1.5.

94. As Sun/Oracle explains it, "Java technology is both a programming language and a platform. The Java programming language is a high-level object-oriented language that has a particular syntax and style. A Java platform is a particular environment in which Java

programming language applications run." OAGOOGLE0024805825 at -34 (Oct. 4, 2006); OAGOOGLE0025057076 at -88 (May 29, 2009); Your First Cup: An Introduction to the Java™ EE Platform at 11 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.

95. As explained further below, *see* ¶¶ 157-161, the Java programming language is free and open for all to use. Sun had for years encouraged its widespread adoption throughout the academic community, both in high school advanced placement courses and in colleges and universities. One of Oracle's corporate representatives admitted that encouraging people to write in the Java programming language was part of Oracle's corporate strategy. Smith Dep. Tr. at 30:21-31:4. It did so in part by making the language, documentation, and API implementations available for free in order to encourage the language to be taught to students. I am aware of Sun's efforts to spread the Java language worldwide through my own experience at Duke. I further was present in the courtroom during the first trial when Sun's former CEO, Jonathan Schwartz, testified that it was "critically important" for Sun to spread Java to "high schools and universities around the world." RT 1958:5-20. Mr. Schwartz explained that Sun "literally went across the world and tried to help universities set up academic curriculum, tried to help them create courseware, tried to give them whatever technology was necessary to aid the students to learn Java." *Id.* Sun's expectation was that after those students graduated "they would go to work for a big company that would become a customer, or they would go off and start a whole new company based on Java." *Id.*

96. Like any high-level programming language, the syntax and style of the Java programming language cannot, as a general matter, be varied. For example, a statement adding

two numbers can only be written in certain ways, and the language requires specific and precise key words to express such things as variable types (e.g., integers, strings, or booleans) and more complex object types such as dates or database queries. In addition, the Java programming language, like many programming languages, employs key words and operators (such as plus and minus symbols) that can only be used for specific purposes and in specific ways; using them for other purposes will cause a program to fail to function correctly. As a result, much of the structure and appearance of code written in the Java programming language is dictated by these functional considerations.

97. As for the broader use of the term "Java" to refer to a platform, Sun historically used the term "Java Programming Language Platform" to describe three platforms: Java Platform, Standard Edition (Java SE); Java Platform, Enterprise Edition (Java EE); and Java Platform, Micro Edition (Java ME). OAGOOGLE0024805825 at -34 (Oct. 4, 2006); OAGOOGLE0025057076 at -88 (May 29, 2009); OAGOOGLE0016828014 at -14. More recently, Sun/Oracle has added a fourth platform called JavaFX. Your First Cup: An Introduction to the Java™ EE Platform (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.

98. As suggested by their names, these different platforms target different programmers, uses, and contexts, each designed to accomplish different things in different environments. That said, the different platforms do share similar structural characteristics as a platform: "All Java platforms consist of a Java Virtual Machine (VM) and an application programming interface (API)." OAGOOGLE0024805825 at -34 (Oct. 4, 2006); OAGOOGLE0025057076 at -88 (May 29, 2009); Your First Cup: An Introduction to the Java™

EE Platform at 11 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. I discuss each platform in further detail below.

1. Java Standard Edition (SE)

99. The Java programming language is inextricably intertwined with the use of the underlying class libraries as presented by the corresponding API. As Sun/Oracle explains it:

> *When most people think of the Java programming language, they think of the Java SE API. Java SE's API provides the core functionality of the Java programming language.* It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.
>
> In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java technology applications.[6]

OAGOOGLE0024805825 at -35 (Oct. 4, 2006); OAGOOGLE0025057076 at -89 (May 29, 2009); Your First Cup: An Introduction to the Java™ EE Platform at 12 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html (emphasis added).

100. Sun characterized Java SE 5.0 as "the premier platform for rapidly developing and deploying secure, portable applications that run on server and desktop systems spanning most operating systems." See http://www.oracle.com/technetwork/java/javase/index-jsp-

[6] Notably, this is a third potential definition of "Java." That is, I understand that Sun/Oracle has also used the term "Java platform" to include a set of core libraries that facilitates the development of applications for the Java platform by providing basic system or language functionalities; a program known as a "compiler" that creates the "bytecode" in which Java programs are executed; and a virtual machine that executes the bytecode. This platoform is sometimes referred to as the Java Runtime Environment, or JRE.

135232.html; *see also* TX 1077 at 3 (noting that the "J2SE™ 5.0 Getting Started Kit contains everything you need to rapidly develop and deploy secure, portable applications that run on server and desktop systems spanning most operating systems.").[7]

101. Desktop and server environments provide more power, memory, clock speed, cooling, etc. than mobile environments, but less than enterprise servers. The Java SE API reflects this fact by providing functionality targeted at this type of environment. For example, Java SE provides APIs for small-grained client/server programs via remote method invocation, or RMI, in the java.rmi packages—this allows Java classes to communicate from within the framework of the Java Virtual Machine rather than using other software protocols. Java SE also provides many packages for developing user interfaces, e.g., java.awt and javax.swing (which are not part of the Android platform). Java SE further provides for image input/output in the javax.imageio packages; for naming services such as identifying printers, LDAP, and other services in javax.naming and javax.print packages; and for several other services that either typically are not part of a mobile platform.

2. Java Enterprise Edition (EE)

102. Java EE extends the functionality of Java SE for the more robust enterprise environment. As Sun/Oracle explains: "The Java EE platform is built on top of the Java SE platform. The Java EE platform provides an API and runtime environment for developing and

[7] Today Oracle describes Java SE as a platform that "lets you develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments." http://www.oracle.com/technetwork/java/javase/overview/index.html. An "embedded" environment is a computer system that has a special function and that is part of a larger mechanical or electrical system. For example, modern automobiles have complex computer systems that control things like fuel injection and emission controls. Such a computer system is "embedded" within the car, and is referred to as an "embedded" system. Oracle includes pictures and a list of such embedded systems, which include ATMs, parking meters and point-of-sale devices, on its website. *See* https://docs.oracle.com/javase/8/embedded/develop-apps-platforms/overview.htm. But back at the time of Android's development, Sun distinguished "Java SE" from "Java SE embedded." TX 2906 ("Related platforms and offerings such as Java EE, Java ME, Java SE embedded, and Java Real-Time offerings are not included with this offering, and require separate agreements.").

running large-scale, multi-tiered, scalable, reliable, and secure network applications." OAGOOGLE0024805825 at -35 (Oct. 4, 2006); OAGOOGLE0025057076 at -89 (May 29, 2009); Your First Cup: An Introduction to the Java™ EE Platform at 12 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.

103. Unlike Java SE, Java EE is meant for server environments that provide more in the way of computing resources than typical desktop and embedded environments. Befitting this more complex environment, Java EE provides additional APIs on top of those provided by Java SE. The Java EE platform adds roughly 100 packages to a set of packages that are part of Java SE. These packages provide services for interacting with programs over the web, for client/server programs running over networks, for processing transactions, for sending or receiving email, and for other services and processes that require large-scale, networked programs.

3. Java Micro Edition (ME)

104. Befitting its name, Java ME is relatively small and provides only a subset of the Java SE APIs. As Oracle explains: "The Java ME platform provides an API and a small-footprint virtual machine for running Java programming language applications on small devices, like mobile phones. The API is a subset of the Java SE API, along with special class libraries useful for small device application development." OAGOOGLE0024805825 at -35 (Oct. 4, 2006); OAGOOGLE0025057076 at -89 (May 29, 2009); Your First Cup: An Introduction to the Java™ EE Platform at 12 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html.

105. Notably, Java ME includes only a very limited number of packages (~10, depending on the profile) as compared to Java SE (which has ~160 APIs in Java 1.5/5, and roughly 200 APIs in Java 7).

106. Although it was used in some mobile phones, Java ME never provided a full-stack smartphone platform for mobile phone and/or application developers. This is addressed in more detail below.

D. The 37 Java API Packages

107. I understand that Oracle is claiming that Google's implementation of the Java API specifications for the following packages does not constitute fair use:

java.awt.font

java.beans

java.io

java.lang

java.lang.annotation

java.lang.ref

java.lang.reflect

java.math

java.net

java.nio

java.nio.channels

java.nio.channels.spi

java.nio.charset

java.nio.charset.spi

java.security

java.security.acl

java.security.cert

java.security.interfaces

java.security.spec

java.sql

java.text

java.util

java.util.jar

java.util.logging

java.util.prefs

java.util.regex

java.util.zip

javax.crypto

javax.crypto.interfaces

javax.crypto.spec

javax.net

javax.net.ssl

javax.security.auth

javax.security.auth.callback

javax.security.auth.login

javax.security.auth.x500

javax.security.cert

javax.sql

The original trial focused on the structure, sequence, and operation (SSO) of these 37 API packages by way of the similar method declarations in the Android Core Libraries. For purposes of this report, I generally treat the SSO and method declarations jointly, and my opinions apply to them equally regardless whether they are considered together or independently.

108. I also understand that Oracle may contend that other Google implementations of Java APIs, method declarations, and/or source code is also infringing, including implementations found in later versions of Android or extensions of the platform into other areas (including wearables, TVs, and cars). Because I have not seen a corresponding expert analysis, I reserve judgment as to whether such alleged uses would be infringing or instead qualify as fair use, and will address that in either or both of my rebuttal and reply reports.

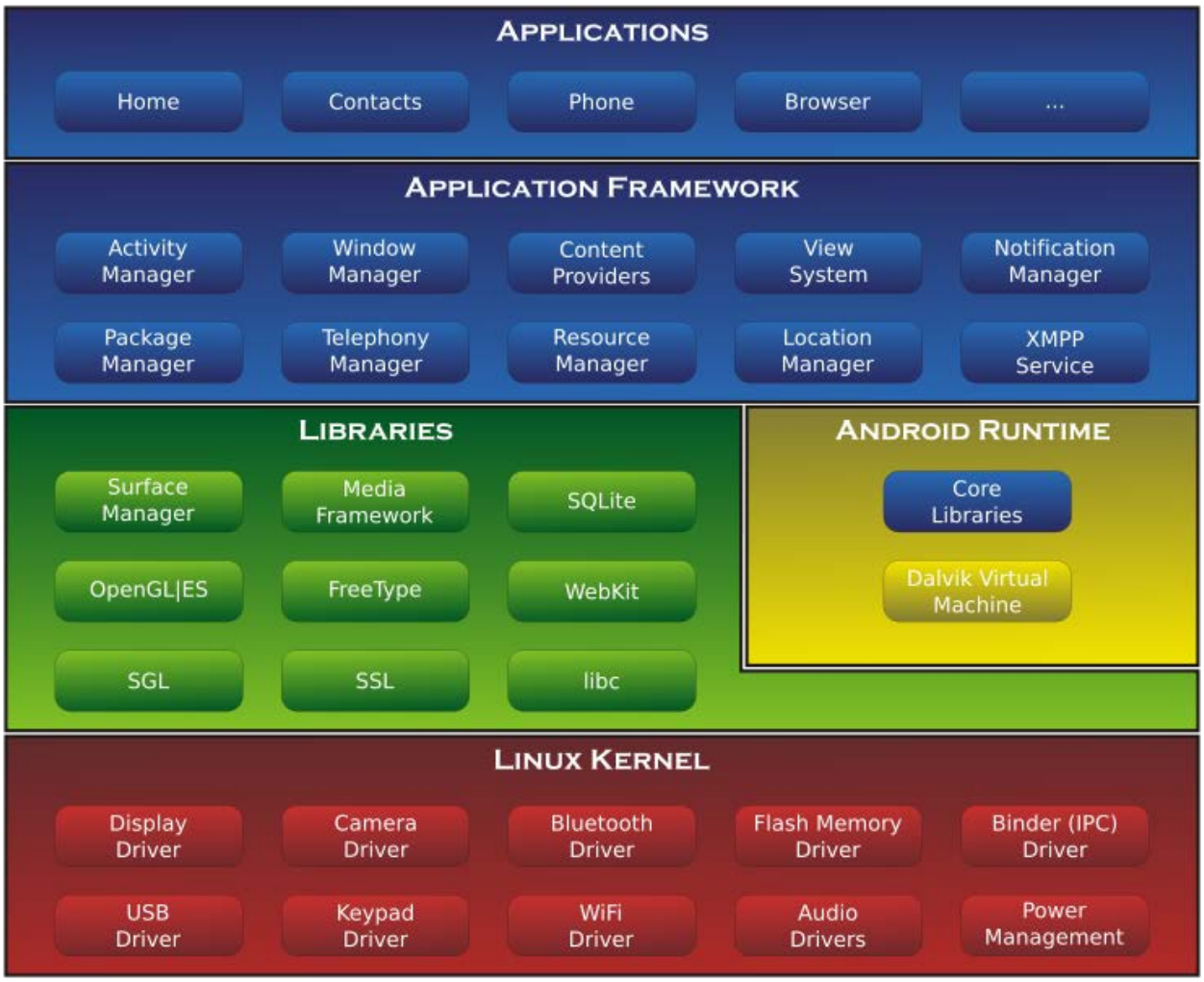
V. ANDROID

A. Android Platform

109. The Android platform is an open and free software stack that includes an operating system, middleware and also key applications for use on mobile devices, including smartphones. This report uses the term Android to encompass all versions of Android published by the Android Open Source Project—basically versions 1.0 through 6.0. *See* https://source.android.com/source/build-numbers.html (listing codenames, versions, and build numbers).[8] That is, my opinions are the same with respect the various different versions.

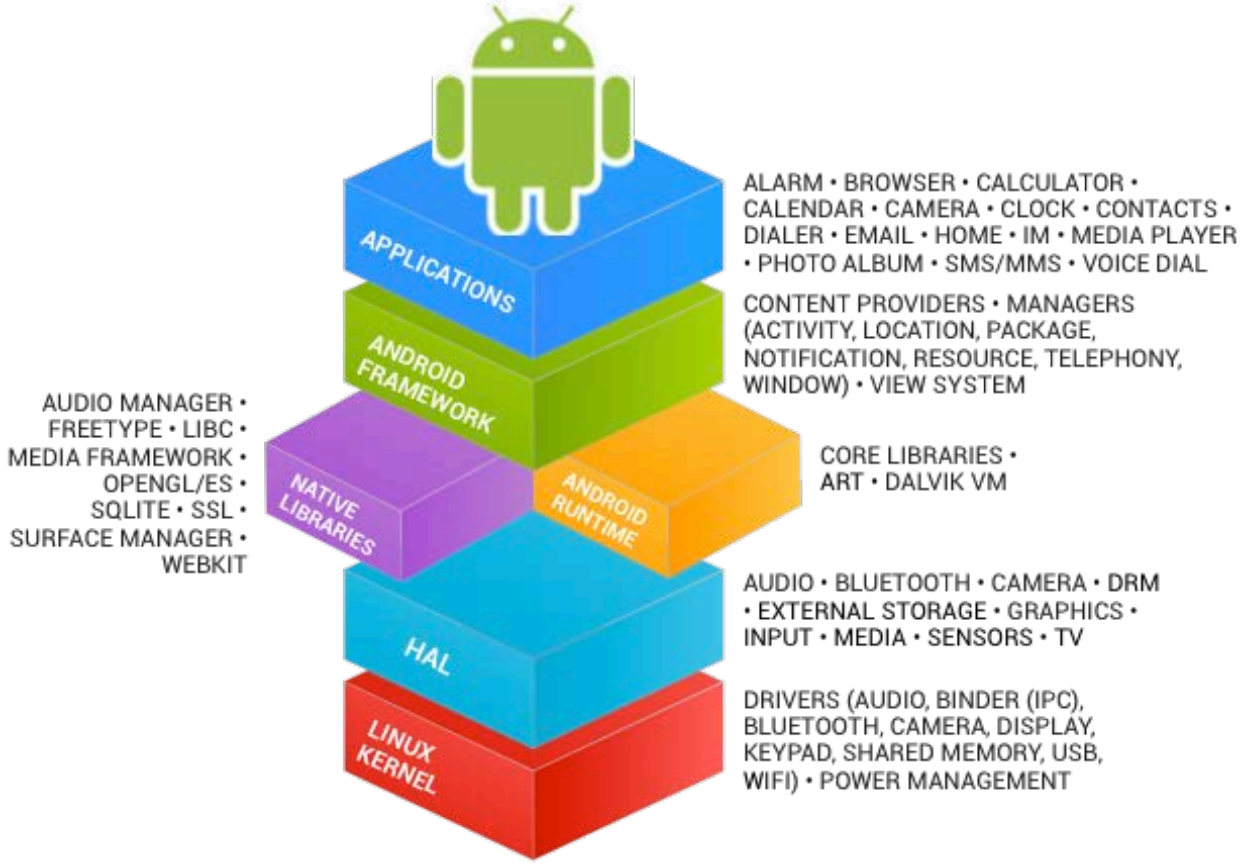
110. The stack is comprised of the layers depicted in the follow graphic, which I will explain in more detail below:

[8] As noted above, though, this report does not address Android Wear, Android Auto, and Android TV, for which I have not yet seen any infringement analysis.



See Android Anatomy and Physiology (*available at* https://sites.google.com/site/io/anatomy--physiology-of-an-android/Android-Anatomy-GoogleIO.pdf?attredirects=0).

111. Another, more recent graphical representation of the stack is as follows:



The Android Source Code (*available at* http://source.android.com/source/index.html).

1. Linux Kernel

112. At the bottom of the Android software stack sits the Linux Kernel. Android's Linux Kernel is based on Linux version 2.6. Among other things, the Linux Kernel provides preemptive multitasking[9] and low-level core system services such as memory, process and power management. By way of a hardware abstraction layer (HAL), the Linux Kernel also provides a network stack and device drivers for hardware such as the device display, Wi-Fi, and audio.

[9] "Preemptive multitasking" is the ability of an operating system to execute multiple programs so that they appear to be executing simultaneously. The operating system uses criteria and responds to external events to transfer the CPU and other system resources between the programs so quickly that to the user the programs appear to execute simultaneously.

113. Notably, the Linux Kernel is licensed under GNU's General Public License with system exceptions, an open source license that allows Google to freely license and freely distribute the Linux Kernel as part of Android.

2. Dalvik Virtual Machine / Android Runtime

114. Google developed its own virtual machine—called Dalvik VM—to run applications. A virtual machine is an "interpreter" that interprets incoming "bytecode" and translates that code into the appropriate native code for the device on which it is running.

115. The Dalvik VM was specifically designed with mobile environments in mind, and was therefore more efficient than a standard Java virtual machine. For example, to execute within a Dalvik VM, an application written in the Java programming language must be compiled and transformed from standard Java class files to the Dalvik executable (.dex) format. That format has a 50% smaller memory footprint than standard Java bytecode.

116. In Android 4.4, Google introduced the Android Runtime (aka, "ART") as an alternative to the Dalvik VM. As of Android 5.0 "Lollipop," ART replaced the Dalvik VM altogether.

3. Core Libraries

117. The Core Libraries fall into three main categories: (1) libraries specific to the Dalvik VM/Android Runtime; (2) Java class libraries; and (3) Android class libraries.

118. The Dalvik VM/Android Runtime-specific libraries relate to the runtime translation of bytecode by the virtual machine. So far as I'm aware, there are no accused-infringing method declarations or APIs in this category of the Core Libraries.

119. The Java class libraries constitute those Java API packages that Google implemented. In total, they include 51 API packages from Java SE 1.5, of which the 37 API packages listed above are accused of infringing. (As discussed in detail below, these API

packages provide functionality that is closely tied to using the Java programming language for a mobile platform.)

120. In developing Android, Google created an additional 112 API packages. *See* Android Package Index (*available at* http://developer.android.com/reference/packages.html). Many of these packages are particularly attuned to a mobile/smartphone environment. For example, Android provides a set of APIs for cameras (in android.hardware) (*see* http://developer.android.com/reference/android/hardware/package-summary.html), location awareness/GPS (in android.location) (*see* http://developer.android.com/reference/android/location/package-summary.html), for playing and recording media files, including both audio and video (in android.media) (*see* http://developer.android.com/reference/android/media/package-summary.html), and telephony (in the android.telephony packages) (*see* http://developer.android.com/reference/android/telephony/package-summary.html). In addition, Android provides a host of features to support smartphone display and input capabilities (in the android.view classes) (*see* http://developer.android.com/reference/android/view/package-summary.html). *See also* Android API Guides (*available at* http://developer.android.com/guide/index.html).

121. A summary of other key Android libraries is as follows:

- **android.app:** provides access to the application model for Android applications.
- **android.content:** facilitates content access, publishing, and messaging between applications and application components.
- **android.database:** used to access data published by content providers and includes SQLite database management classes.
- **android.graphics:** low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.

- **android.opengl:** Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os:** provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.net:** set of APIs providing access to the network stack. Includes android.net.wifi, which provides access to the device's wireless stack.
- **android.provider:** set of convenience classes that provide access to standard Android content provider databases such as those maintained by the calendar and contact applications.
- **android.text:** set of APIs for rendering and manipulating text on a device display.
- **android.util:** set of utility classes for performing tasks such as string and number conversion, XML handling and date and time manipulation.
- **android.view**: fundamental building blocks of application user interfaces.
- **android.widget**: collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit**: set of classes intended to allow web-browsing capabilities to be built into applications.

See An Overview of the Android Architecture (*available at* http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture).

122. Notably, these core libraries do not always perform the actual "work"; rather, many of them are Java "wrappers" around a set of C/C++ based libraries. Android developers access these C/C++ libraries through the Java-based Android core library APIs. The Java APIs are still used by programmers when they write Java programs that make calls like Math.sqrt(25.0). However, rather than implementing the code to calculate square roots in Java, a library written or tested in C or C++ to calculate square roots would be called. No Java code would appear in the body of the Math.sqrt method in the implementation code. Instead, the Math.sqrt method would be declared as native, so that the C/C++ code that implements the

square root functionality will be called to return 5.0 when a programmer writes, in Java, Math.sqrt(25.0). For example, Java and other languages often make it possible to call code libraries "natively," that is code that runs directly on a chip's CPU rather than in a virtual machine. Since C and C++ can be compiled to run directly by the CPU, this compiled code can be called by Java implementations of some APIs. For example, math libraries are sometimes written in C or C++ and compiled into code that runs directly on the CPU. Thus, the Java 1.5 and Android libraries both call native methods when the Math.sqrt API is invoked. If and when direct access to these libraries is needed, developers can use the Android Native Development Kit (NDK). The NDK allows developers to call the native methods of non-Java programming languages (*e.g.*, C/C++) from within Java code using the Java Native Interface (JNI). This means that in many cases, applications need not be written in the Java programming language; indeed, many of the most popular gaming applications are in fact written in languages such as C++. The JNI is not specific to Android, but works on the Java SE Platform as well. *See* http://docs.oracle.com/javase/7/docs/technotes/guides/jni/.

4. Libraries

123. The other libraries in Android come from a variety of different open-source and other projects.

124. For example, the WebKit library is based on open source WebKit browser code from http://webkit.org. Among other things, WebKit renders pages in full (desktop) view; provides full CSS, Javascript, DOM, and AJAX support; and supports single-column and adaptive view rendering. *See* Android Anatomy and Physiology (*available at* https://sites.google.com/site/io/anatomy--physiology-of-an-android/Android-Anatomy-GoogleIO.pdf?attredirects=0).

125. In addition the "non-core" libraries provide support for database management and querying (SQLite), media playback (OpenCORE), secure communications (SSL), and various other features that are important to the Android framework and end-user experience.

5. Application Framework

126. Collectively, the Android Application Framework provides services for running and managing Android applications. Among other things, it implements the idea that Android applications are constructed from components that can be reused and/or interconnected with other applications/components.

127. The Android framework includes the following key services:

- **Activity Manager:** controls all aspects of the application lifecycle and activity stack.
- **Content Providers:** allows applications to publish and share data with other applications.
- **Resource Manager:** provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager:** allows applications to display alerts and notifications to the user.
- **View System:** an extensible set of views used to create application user interfaces.
- **Package Manager:** the system by which applications are able to find out information about other applications currently installed on the device.
- **Telephony Manager:** provides information to the application about the telephony services available on the device such as status and subscriber information.
- **Location Manager:** provides access to the location services allowing an application to receive updates about location changes.

See An Overview of the Android Architecture (*available at* http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture).

6. Applications

128. Applications sit at the top of the Android "stack." They include applications provided with Android (*e.g.,* a web browser, email applications), applications pre-loaded by an OEM, and applications installed by a user (e.g., from the Google Play store). Some applications are written in the Java programming language, while others are written in non-Java programming languages (*e.g.*, C/C++).

B. Android Distribution

129. The Android code is available for free download from http://source.android.com/. As explained on the website, "Android is an open source software stack for a wide range of mobile devices and a corresponding open source project led by Google. This site offers the information and source code you need to create custom variants of the Android stack, port devices and accessories to the Android platform, and ensure your devices meet compatibility requirements."

130. Android provides the Android code base subject to a variety of open-source licenses. *See* https://source.android.com/source/licenses.html (describing the different licenses applicable to Android). "The preferred license for the Android Open Source Project is the Apache Software License, Version 2.0 ("Apache 2.0"), and the majority of the Android software is licensed with Apache 2.0. While the project will strive to adhere to the preferred license, there may be exceptions that will be handled on a case-by-case basis. For example, the Linux kernel patches are under the GPLv2 license with system exceptions, which can be found on kernel.org." *Id.* That said, as noted above, certain components of Android are subject to other licensing terms. For example, the Linux Kernel is licensed under the GNU Public License, version 2.0 ("GPL-2.0 license"). As another example, WebKit is also licensed under the GPL-2.0 license or,

alternatively, a Berkeley Software Distribution ("BSD") open-source license. *See* https://webkit.org/licensing-webkit/.

131. Google also recently released a new implementation of the 37 API packages using the OpenJDK code base, which is licensed under the GPL-2.0 with Classpath Exception license ("GPL-2.0-CE license"). I understand that the other core libraries, including an additional 14 Java SE API packages in which Oracle does not assert infringement, as well as the 112 other Android packages developed by Google, will continue to be licensed under the Apache 2.0 license.

132. In short, Google mixes and matches different open source licenses for different Android components, depending on, among other things, the source of the underlying code and Google's declared open-source licensing preferences.

VI. ANALYSIS OF FAIR USE FACTORS FOR DECLARATIONS IN THE 37 API PACKAGES

A. The Purpose and Character of the Use

133. In my opinion, Google's use of the method declarations in the 37 API packages gave them a new expression, meaning, or message so as to create something new, with a further purpose or different character than Java SE. Google's use of the method declarations in the 37 API packages as part of the larger, mobile-targeted Android platform is substantially different from Java SE, which—as discussed above—was directed to applications for desktop computers and servers (as well as demanding embedded environments). As such, Android's use of the method declarations from the 37 API packages from Java SE was highly transformative because Google made substantial and important changes to the original work.

134. As discussed above, Java SE provides a broad set of APIs, befitting an environment with sufficient power, storage, and computing power. That well describes a

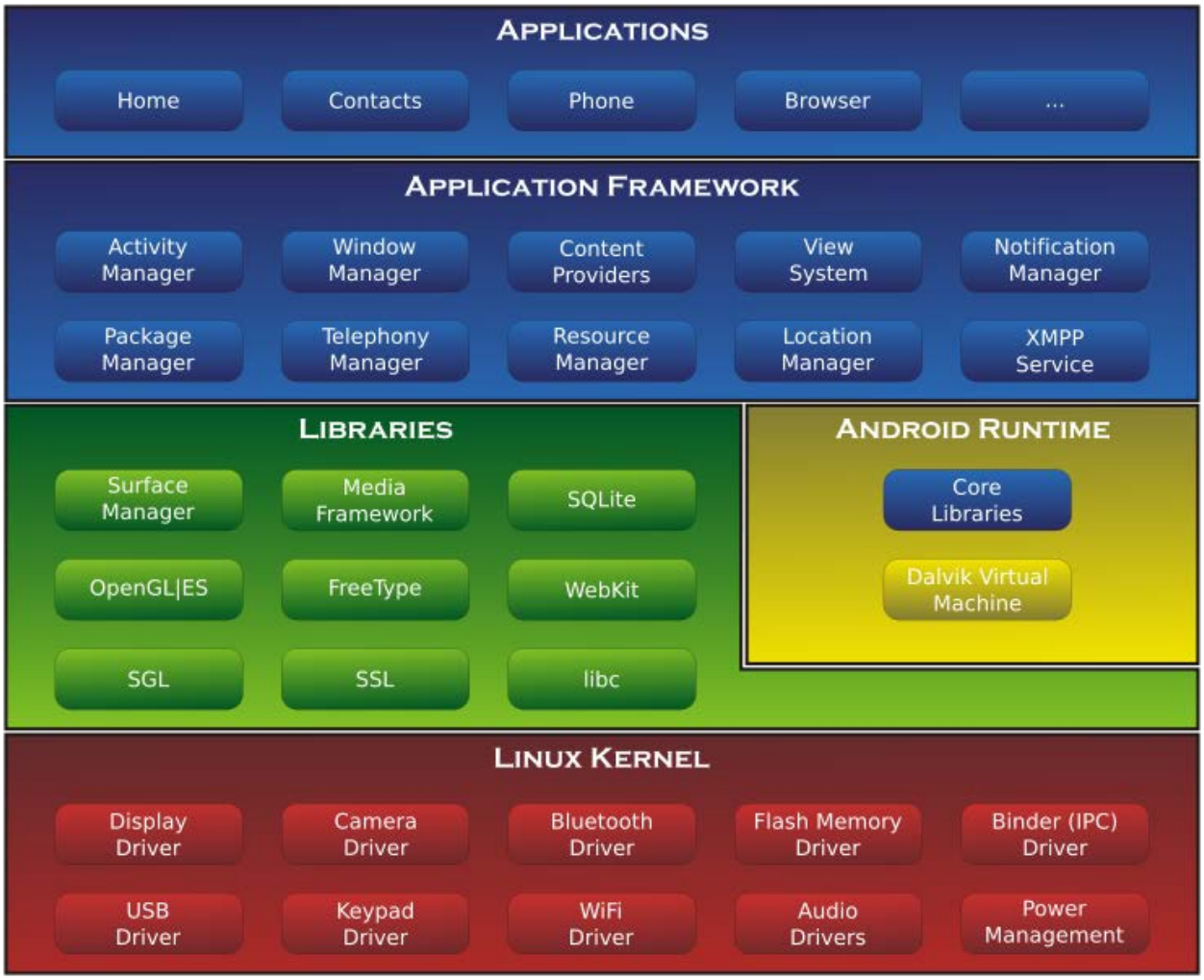
desktop computer or server (as well as common "demanding" embedded systems—e.g., cars, ATMs, and point-of-sale devices), to which Sun/Oracle targeted Java SE (as discussed above). Java ME and EE are also consistent with their respective intended usages (as also discussed above).

135. Android uses the method declarations from the 37 API packages in a very different context from Java SE (as well as from Java EE and Java ME), because Android was designed for use in "smartphones"—mobile devices with touch-screen interfaces and a framework for running applications. Furthermore, in comparison to the environments for which Java SE was tailored—desktops and servers (as well as demanding embedded environments)—mobile devices generally have less power, processor speed, and memory. They also have different input/output characteristics, including, for example, smaller displays with touch-screen capability, and rely on different networking capabilities (*e.g.*, cellular connections). Accordingly, many of the APIs in Java SE (or EE, for that matter) are not well-suited or appropriate for smartphones, whereas the Java ME APIs were not sufficiently robust to support a full smartphone platform. As a result, until Android transformed the 37 Java API packages, they had never been successfully used to support a full-stack smartphone platform.

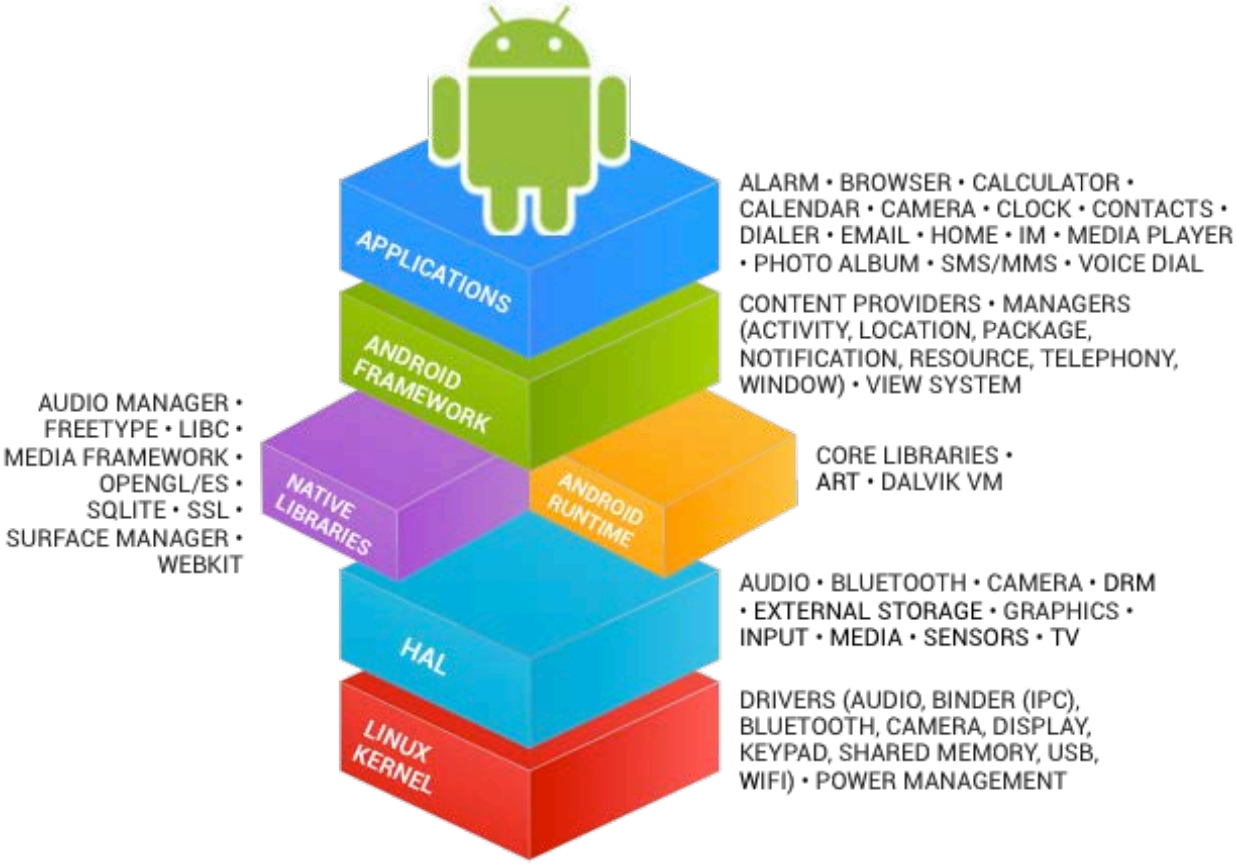
136. As discussed above, Google built a variety of APIs for Android that are specific to smartphone devices (*e.g.*, APIs for cameras, location awareness/GPS, audio/video playback and recording, and display features). *See also generally* Package Index (*available at* http://developer.android.com/reference/packages.html); Android API Guides (*available at* http://developer.android.com/guide/index.html).

137. Furthermore, the 37 Java API packages that have been implemented by Google for Android are a small part of the overall Android system, both in terms of the functionality they

provide and the lines of code involved. The 37 API packages at issue are roughly one-fifth of the Android Runtime Core Libraries, which currently contains 168 API packages. The Core Libraries, in turn, are themselves only a small part of the overall Android architecture—they are the small blue box labeled "core libraries" in the yellow box at the right.



Represented another way, they are simply a part of the orange "Android Runtime" layer.



138. Besides the 37 Java API packages in Android, the platform includes the Dalvik Virtual Machine and/or ART, the Linux kernel, a web browser, and a variety of other libraries and systems that are not part of the Java platform. These other components, when combined with the Android Core Libraries, make for a complete mobile operating system—something substantially different in scope and ability than Java SE.

139. While the diagrams above are not "to scale" (boxes and/or layers of the same size may represent software of different size and complexity), an analysis of the number of lines of source code in the API packages at issue, in the Android Runtime Core Libraries, and in the other components in the diagram suggests that, if anything, the diagram likely over-represents the size of the APIs at issue.

140. Using a Python script SlocCounter.py based on the "sloccount" tool, a commonly-used tool for measuring the size of the source code of large software projects, Android's implementation of the APIs at issue in the "Gingerbread" release constitutes 259,474 lines of code, in 1022 files. This is roughly 1.6% of the size of the entire Android source code, which comprises 57,076 files and 15,347,169 lines of code,[10] and roughly 15% of the 6,340 files and 1,713,087 lines of code[11] in the overall Android (Gingerbread) Runtime Core Libraries. Similarly, implementation of these APIs is a small portion of Oracle's JDK 1.5 implementation of the entire Java API, constituting 315,570 lines of code out of 2,867,712 (11% of the total) and 1001 files out of 9521 (10.5%).

141. Accordingly, Google's additions created something new, with a further purpose and different character, adding new expression, meaning and message. Standing on their own, the method declarations from the 37 API packages do not provide a system for mobile devices such as smartphones. While they provide interfaces for accessing basic functionality that programmers would expect in a mobile device platform, a mobile device platform needs much more than that basic functionality. First, Google also transformed the 37 API packages themselves by way of the different implementations (discussed in detail below), which were designed to make the APIs work for a mobile device (*e.g.*, by making them more compact and efficient). Second, without the *other* Android APIs that Google developed and combined with the 37 Java API packages, Android would not have been able to develop a platform appropriate for mobile devices. Third, Android goes beyond just allowing the use of applications written in

[10] Numbers generated by running SlocCounterTotal.py against a clean copy of Android (obtained following the instructions available here: http://source.android.com/source/downloading.html) and including only lines of code in .h, .c, .cpp, and .java files.
[11] Numbers generated by running SlocCounterTotal.py against the libcore/ and frameworks/base/core/ directories in a clean copy of Android, counting only lines of code in .h, .c, .cpp, and .java files.

the Java programming language; through its Native Development Kit, Android also allows developers to create and run programs in native languages such as C++ on Android devices.

142. Moreover, Google's additions use the method declarations from the 37 API packages for purposes distinct from their original purpose. Google's use does not supersede Java SE, because the object of Java SE was to empower programmers targeting desktops and servers (as well as "demanding" embedded environments). In contrast, as discussed above, Android is designed for mobile devices, which have very different needs.

143. Furthermore, these differences are important because the context influences how or what programmers develop—and therefore what API packages they need. Indeed, applications written for a smartphone are very different than programs for desktop computers. For example, very few people are interested in full-blown word processors on smartphones. Even where there are some overlaps there are important differences. A programmer writing an application for a smartphone needs to keep in mind that the user does not have access to the same hardware one would have for a desktop/laptop system (e.g., a hardware keyboard and/or mouse). In addition, smartphones often come with input systems that are not common on desktops, including accelerometers, touchscreens, and GPS.

144. Notably, Google does not sell the Android operating system. Instead, it makes it available to anyone for free. By publishing the entirety of the Android source code, Google allows anyone to use that source code as is or to modify it, all for no charge. The Android API packages, including the method declarations from the 37 API packages at issue here, are part of that free distribution.

145. How the free and open source Android code has been transformative is reflected in how various companies have adopted and used it. For example, the original Amazon Kindle

was a single-purpose device designed for reading electronic books—often referred to as an e-reader. When Amazon sought to expand the use cases for its Kindle product by creating a tablet device, Amazon started with the Android code base to develop its FireOS that runs its Kindle Fire line of products (first released in 2011).

146. FireOS has been modified from the version of Android made available by Google to emphasize the goods and services that can be purchased through Amazon. Thus, Fire devices are optimized for tasks like watching content streamed from Amazon.

147. FireOS devices also do not have Google applications that come on many Android devices, such as the Google Play store, or Google Maps. Users may be able to load some of these applications onto FireOS devices, but they are not found there by default; instead, the FireOS devices are designed to be part of the Amazon ecosystem. Thus, although Amazon takes advantage of the Android source code made freely available to all by Google, it does so in a very different way than OEMs that use the Android source code to power Android mobile devices.

148. Another example of the transformative nature of the Android source code being used to create a new and different project is the CyanogenMod operating system. CyanogenMod is a free and open source platform that can be installed on Android devices in place of the standard Android firmware. Its creators claim that those who choose to install it will experience better performance than they would get from the standard Android operating system. It can also be used to give a device a different look and feel, due to the availability of a different "launcher" for starting applications on the Android device. CyanogenMod started out as firmware that could be downloaded and substituted for the Android firmware that was run by early Android devices like the HTC G1. CyanogenMod has been continually updated with new releases of Android source code so as to provide an alternative for users of Android devices.

149. As the CyanogenMod project has matured, it has also become available as the default operating system on some mobile devices. For example, the OnePlus One mobile phone shipped with a variant of the CyanogenMod operating system already installed and recently changed to another Android variant named OxygenOS. Other firms shipping mobile devices with CyanogenMod include YU Televentures, Wileyfox, Lenovo, and Smartfren. CyanogenMod thus uses the Android source code that Google makes freely available to create an operating system that can be used in place of Android on an Android device that a consumer already owns, or even be sold directly to a consumer as a substitute for a device running Android.

150. Even Oracle's employees acknowledge that Android is transformational. Terrence Barr, Oracle's Senior Principal Technologist and formerly a Java Evangelist at Sun, testified that one of the purposes of Android was to transform the mobile industry, and that Android was in fact "transformative."[12]

151. Further demonstrating how transformational Android has been, Oracle's OpenJDK project is now being ported to Android. Specifically, Oracle notes that "The goal of this Project is to focus on porting the JDK to popular mobile platforms such as iOS, Android, and Windows." http://openjdk.java.net/projects/mobile/. To that end, Oracle even provides links to the open-source Android project and instructs people to download the NDK and SDK. http://openjdk.java.net/projects/mobile/android.html ("Our JDK 9 Mobile project requires the Android NDK and SDK which can be downloaded from Android developer site.").

152. Oracle/Sun had tried to bring about just such a transformation in the Java APIs by adapting them for use in a smartphone platform, but these projects were all either not funded or ended in failure. In contrast to the many companies successfully using the transformative Android platform today for smartphones, Sun/Oracle employee Craig Gering testified that,

[12] Barr Dep. Tr. at 137:19-23, 138:12-15; *see also id.* at 133:21-134:19, 135:10-12; DX 1368.

despite several different efforts, Sun never brought a full stack mobile operating platform to the market.[13] Mr. Gering also admitted that even projects to put a Sun Java virtual machine onto the Android platform did not result in any product being brought to market.[14]

153. In sum, Android was a transformative use of the 37 API packages in that it created an entirely new smartphone platform—something that Java SE (as well as Java EE and ME) did not provide. As further evidence of Android's transformational use, it enabled the creation of additional varied operating systems and platforms by others.

B. The Nature of the Copyrighted Work

154. Taking into account the legal standards provided to me, and given the nature of the work, it is my opinion that the method declarations in the 37 API packages are largely functional. They are very different from the expression in, for example, a fictional story or even the expression that implements the functionality provided by the 37 API packages.

155. The entire purpose of an API is to allow one program to "interface" with another "application." This interfacing is not a social or creative chat, but a formal, functional command from one program to another: "Do this thing for me, and report back when you are done." The program in command is using the API to operate the underlying program; and the underlying program, likewise, is being operated by means of the API. In fact, it is typically difficult, if not impossible, to operate the underlying system in any way except through an API.

156. Like "Ctrl+P," the print icon, or the steering wheel, APIs provide mechanisms that operate underlying functionality, causing the libraries at issue to perform activities and return the information requested by the programmer.

[13] RT 1912:11-13; *see also* RT 1939:11-12 (Oracle employee Hasan Rizvi admitting that Oracle has never had a full stack on the market for the Java platform).
[14] RT 1915:16-18; RT 1917:15-19.

1. The declarations in the 37 API packages are closely tied to using the Java programming language for a mobile smartphone platform.

157. I understand that there is no dispute in this case that the Java programming language is free and open for all to use. That comports with my understanding based on my expertise, and, in my opinion, is the prevailing understanding of computer scientists.

158. Based on their familiarity with the Java programming language, many software developers are familiar with Java APIs. Sun went to great lengths to encourage developers to learn and use Java. This began when Sun made the Java language, documentation, and API implementation available at no charge in 1996, apparently with the intent to ensure that programmers throughout the industry knew and had internalized the Java language, including these APIs. Sun also worked extensively with educational institutions to help make the Java language a common tool for introductory programming classes.

159. For example, there was active collaboration between Sun and the College Board in promoting Java as the language to be used in the Advanced Placement Computer Science (AP CS) exams developed by the College Board. These changes began just as the exam was switched from Pascal to C++ in 1999. Java was too large a language to be taught completely, so a subset of the classes and methods were identified as being an important part of teaching computer science. This led to the development of a group of classes that were part of the AP CS program and that mirrored the Java API exactly, e.g., instead of being part of the java.lang packages, the AP CS program identified and used ap.java.lang with a corresponding documentation as part of the subset.

160. At a similar time, Sun collaborated with the BlueJ group headed by Michael Kolling to develop an IDE (Integrated Development Environment) for novice programmers that was simple, but that used best practices that were part of Sun's official NetBeans IDE. The

development of BlueJ led to a worldwide adoption of BlueJ in many colleges and high schools that continues today with the use of BlueJ and its derivative GreenFoot which sees widespread adoption.

161. In part as a result of Sun's efforts, I taught Java to my students for many years, and continue to do so. This effort created a large base of programmers who had learned the Java language and APIs. In fact, on its website, Oracle claims that Java is the number one programming language. *See* Oracle Java Homepage (*available at* https://www.oracle.com/java/index.html) (touting that "Java is the world's #1 programming language"). Sun/Oracle's decision to open source Java SE in the OpenJDK project is yet another example of trying to spread Java as far and wide as possible.

162. When Google chose the Java programming language for Android, certain things naturally flowed from that decision—including using method declarations from the Java APIs to the extent that functionality is incorporated into the new platform. That is, to the extent that Google implemented functionality that was the same as functionality already used by Java programmers, programmers would have expected Android to use the same method declarations. Had Google changed those well-known method declarations, that would have merely confused Java programmers who were accustomed to using those method declarations.

163. As Oracle itself explains that "When most people think of the Java programming language, they think of the Java SE API." OAGOOGLE0024805825 at -35 (Oct. 4, 2006); OAGOOGLE0025057076 at -89 (May 29, 2009); Your First Cup: An Introduction to the Java™ EE Platform at 12 (April 2012) (*available at* http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf); http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. Similarly, Oracle's documentation

describes the classes in the java.lang package (one of the API packages at issue) as "fundamental to the design of the Java programming language." *See* http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/package-summary.html; *see also* The Java Application Programming Interface, Volume 1: Core Packages (describing the core packages, including java.lang, java.io, and java.net as "the general-purpose libraries fundamental to every Java program").

164. Along those lines, Dr. Mark Reinhold, Oracle's chief architect of the Java Platform Group, admitted at trial that "[a]t the very least you need to use" the APIs and class libraries "that are tightly related to the Java programming language" in order to program in the Java programming language. RT 684:16-20; *see also* TX 1062 (containing the list of classes and interfaces that Dr. Reinhold concedes are needed for use of the Java programming language); RT 1274:16-19 (Oracle expert Dr. John Mitchell conceding that some APIs "are essential to the language").

165. Similarly, Donald Smith, Oracle's senior director of project management, testified that the Java APIs are fundamental part of the Java language:

> Q: So do you understand the Java language to include the APIs?
>
> A: Yes. I mean, the APIs are a critical part of the Java language.
>
> Q: Would you say that's true for the APIs that are at issue in the case?
>
> A: Yes, those APIs are a fundamental part of what makes Java Java -- what makes a developer recognize Java.

Smith Dep. Tr. at 22:16-23:2 (objections omitted). He further testified that the language and the APIs are "not separable. It's all defined together under the same specification." *Id.* at 24:5-8.

166. Terrence Barr, another Oracle employee, provided similar testimony:

Q: So does the Java language depend on the Java APIs in a technical sense?

A: The Java language depends on Java APIs as part of the entire Java platform. It's important to understand that the Java platform is a well-defined set of functionality that needs to be consistent and is licensed and subject to these compliance requirements. And so the Java language is viewed as part of that platform.

Barr Dep. Tr. at 57:8-19 (objection omitted).

167. Other industry participants see it the same way. As John Duimovich of IBM testified:

Q: I think you said before that the libraries that were contributed by IBM to the Apache Harmony Project were core libraries; is that right?

A: Correct.

Q What makes them core libraries?

A: What makes them core is that they contain classes that are fundamental in the Java language.

. . .

Q: Why is it important for a developer writing software in the Java programming language to be able to use these core classes?

A: They're necessary to write Java code.

Duimovich Dep. Tr. at 60:24-62:7 (objections omitted).

168. In short, asking a software developer to program in Java without using the corresponding APIs is asking that programmer to ignore aspects of the language that Oracle itself has called "fundamental." This is like asking someone to write in the English language but then denying them the ability to use common, well-known English words. In short, it is a bad idea to change method declarations when you are supporting the same functionality on your platform.

169. Conversely, if you are going to use new or different method declarations, you should have a good reason for doing so. For example, where the functionality offered by Android is different from that offered by Java SE—*e.g.*, the Android user interface uses a touch screen instead of a mouse and keyboard and is designed for a small screen—it made sense for Google to create new method declarations. The existing method declarations were ill-adapted to this new purpose, and trying to extend them to encompass a new purpose would have resulted in method declarations that would not serve that purpose well.

170. Finally, it would not make sense to include method declarations for APIs that you are not supporting on your platform, even if those method declarations are well known. Accordingly, Google would have had no reason to include method declarations for numerous Java SE APIs where it decided that the Android platform did not need the associated functionality.

171. In my opinion, had Google implemented none of the method declarations for the 37 API packages in Android, programmers would not have even recognized the platform as implementing the Java programming language. It is my further opinion that using the method declarations from the 37 API packages was reasonable for ensuring basic compatibility with the conventions of the Java programming language and expectations of programmers for a mobile platform. For example, the java.net package contains functionality relating to networking, and every modern mobile platform has networking functionality. Furthermore, had Google implemented different method declarations for the 37 API packages, programmers who had grown accustomed to programming in the Java language would have found it cumbersome to use the Java language when programming for the Android platform. And conversely, programmers

who would grow accustomed to programming for the Android platform would find it cumbersome to use the Java language elsewhere.

172. I have also considered the testimony of Dan Bornstein about Google's choice of which Java APIs to implement. That testimony comports with my own opinion that Google reasonably concluded that programmers for a mobile platform would expect to have available to them the 37 API packages.

173. It is worth emphasizing that reuse of the method declarations of the 37 API packages did not include use of Oracle's underlying program logic implementing those methods; to the contrary, Google wrote its own implementations.

2. The rules and naming conventions in the Java Language Specification constrain the choices for Java API element names and how they are organized.

174. Java API element names frequently repeat certain key terms and patterns, following mechanical rules laid out in the Java Language Specification and elaborated over time by practice. Applying the legal standards provided to me, it is my opinion that because the Java API element names follow these mechanical rules, they are only entitled to weak copyright protection.

175. Specifically, the Java Language Specification provides conventions for structure and naming, stating, for example, that "[m]ethod names should be verbs or verb phrases, in mixed case, with the first letter lowercase and the first letter of any subsequent words capitalized." Similarly, names of class types are to be "descriptive nouns or noun phrases." Java Language Specification, First Edition, Section 6.8 "Naming Conventions." Both of these rules are followed by all the examples shown in this report, except for those methods that are drawn from older programming languages (like "sqrt").

176. Additional word patterns crop up repeatedly throughout the Java APIs. "InputStream" and "ChangeEvent" are two examples affecting a few dozen names, but others go much further. For example, the Java Language Specification conventions for method names state that methods that return the value of a variable should start with "get," and method names that set the value of a variable should start with "set." Other rules require specific methods to be found in many classes, such as "hashCode," "toString," and "equals". In Oracle's implementation of Java 1.5, nearly one-third of the method names at issue (2,456 of the 6,821 methods) are determined by these rules, including roughly 2,000 that begin with either "get" or "set" and 142 named simply "equals." Getting or setting a class member's value is a common operation for programmers, and thus it makes sense that many Java classes would include "get" and "set" methods. To avoid confusion, it makes sense to follow these naming conventions for those methods. Because Java is an object-oriented language, the case of "equals" is even stronger. Code would not work if classes used a method name other than "equals" to check for equality. If a method for checking equality were instead named, for example, "sameAs," then code that tested for equality would not work. Because Java is an object-oriented language, the specific rules mentioned above require that some names, like "equals," be used across all classes. These constraints yield the resulting names ("equals," "set," "get," etc.)—which are functional rather than creative choices. The use of "set" and "get" is because of convention and expectation rather than because of the rules required by object-orientation.

177. An additional 1,841 method names were single words, like "run" or "add." The remaining 2,524 methods are not long or complicated; they are, on average, only 2.339 words "long" (counting a method name like locateURL as two words and findBestMatch as three words).

178. The results from using Java 1.6 are similar: 2,600 of 7,487 methods (including more than 2,000 get/set methods and 150 methods named "equals") are determined by the aforementioned rules and guidelines of the Java Language Specification. In Java 1.6 there are 2,001 method names that are single words, and of the 2,886 other method names, the average word length is 2.343.

179. Following these mechanical rules and seeking to create consistency reduce the amount of creativity and work necessary to write the API, and, more importantly, reduces the amount of work necessary to learn and memorize the API. As a result, any good API design will have naming rules like the Java API naming rules contained in the Java Language Specification, which result in names that are functional and primarily dictated by efficiency concerns.

180. Importantly, the foregoing points apply not only to names in the method declarations, but also how they are organized. Just as the name must be tied to functionality, organizing APIs into related groupings must also reflect underlying functionality so that programmers can discover and use them efficiently. For example, methods related to security, such as AccessController.checkPermission and Signature.sign, are most sensibly organized into packages primarily related to security—java.security and javax.security.

181. These practical requirements restrict the packages into which any class could sensibly be placed, and similarly restrict the classes into which any method could sensibly be placed. The fourth column in the tables below (at ¶ 195) demonstrates how and why various classes fit into their respective packages, and various methods fit into their respective classes. For example, ZipInputStream is a class that allows creation of an "input stream" that can read from a compressed file (known as a .zip file). Because it relates to .zip files, developers will

look for it with other classes and methods related to .zip files, and so it is grouped with them in an appropriately named subpackage called java.util.zip.

3. The names of the 37 API packages are closely tied to their function.

182. Because the names of the 37 API packages are short, fragmentary, and descriptive, they reflect only marginal creativity.

183. It is my understanding that the following API packages (and, in some cases, certain sub-packages) are at issue in the case. In each case, the name of the package and the basic organization of the classes and methods within each package are merely descriptive of the functionalities in those packages.

java.lang

The java.lang package and its sub-packages java.lang.ref, java.lang.reflect, and java.lang.annotation are part of a group of classes that facilitate interacting with and programming related to the Java language. The package name ("lang") and contents of the classes and methods in this package reflects this emphasis on the core Java language.

java.math

The java.math package provides the programmer with access to classes that facilitate arbitrary precision arithmetic with integers, *e.g.*, integers with no upper or lower limit. The package name ("math") and contents of the classes in this package reflect this underlying functionality.

java.net

The java.net package, and its extension javax.net and sub-package javax.net.ssl, provide classes for the programmer to implement network connections at both a low and high

level. The package name ("net") is short for "network" and therefore reflects this underlying functionality.

java.io and java.nio

The java.io and java.nio packages group together classes for dealing with input and output. Input and output are called "I/O" in long-standing programmer jargon, explaining the name of the io package. (The nio package is so-named because it was an attempt to present a "new" io (nio) package.) The java.nio hierarchy of classes also contains the sub-packages java.nio.channels, java.nio.channels.spi, java.nio.charset java.nio.charset.spi. The nio package provides classes that facilitate more efficient (faster) input and output. The nio package are designed to interact with each particular operating system's efficient I/O mechanisms, so that the Java programmer can use the nio classes knowing that they will likely be faster than the java.io classes that were not originally designed for efficiency.

java.security and javax.security

The java.security package, its sub-packages java.security.acl, java.security.cert, java.security.interfaces and java.security.spec, and its extensions, javax.security.auth, javax.security.auth.callback, javax.security.auth.login, javax.security.auth.x500, javax.security.cert, provide classes and functionality related to security, as the names suggest.

java.sql

The java.sql package, and its extension, javax.sql, facilitates interacting with relational databases or data in a format similar to that defined in such a database. These packages

are based on the "SQL" standard (Structured Query Language), a standard named and defined in the late 1970s and early 1980s, and the name of the packages ("sql") reflects the name of this standard.

java.text

The java.text package facilitates writing software to handle text, dates, numbers, and messages in a format that is independent of a particular natural language—allowing programmers to cope more easily with languages other than English.

java.util

The java.util packages provides utilities and collection classes. Its subpackages java.util.logging, java.util.jar, java.util.prefs, java.util.regex, and java.util.zip provide utilities that are more specialized, *e.g.*, to deal with logging, archives (jar files), user preferences, regular expressions, and zipped or compressed files, respectively. These functionalities are diverse, but "utilities" are a traditional name for small, single-purpose tools in the computing world, and so grouping these together under the name "util" is a straightforward mapping of functionality to traditional naming.

javax.crypto

The javax.crypto package and its subpackages javax.crypto.interfaces and javax.crypto.spec provide classes to write code that adheres to cryptographic protocols.

In some cases, a given package may require the functionality of another package in order to function correctly, much like the upper floors of a building need the lower floors of the building to remain standing. The final two packages listed below, while not themselves basic to the

functionality of modern operating systems, must be present in order for the previously listed packages to operate correctly and provide their complete, intended functionality to users:

> **java.awt.font**
>
> The package java.awt.font allows programmers to interact with low-level font information.
>
> **java.beans**
>
> The java.beans package facilitates software interaction with JavaBeans—traditionally viewed as a reusable software component conforming to specific conventions so that the component can be manipulated with visual and graphical tools.

184. Because these names all describe specific functionalities, and are limited by design rules to short, fragmentary words and phrases, there is limited creativity in the package names.

185. Had the authors of the declarations so desired, they could have chosen far more creative names. For example, "java.crypto" could have been called "java.supersecret" or "java.spygames." Instead of "java.security," they could have chosen "java.letskeepthingsafe." Indeed, they could have abandoned the choice of "java" as the prefix for these package names. Those would have been more creative choices—but they would have greatly impaired the functionality of the API packages, by making the APIs hard to remember and use. This is contrary to the very purpose of having APIs.

186. When a programmer uses an API as part of their code, for example, to sort prices, they want to focus on how best to express their sorting algorithm. They do not want to be distracted by the creative whims of an API designer who chose to put mathematical methods

commonly used by Java programmers in a package called "addingsubstractingandothernerdystuff" instead of the far more garden-variety choice "java.math."

4. The names of the classes and methods are also closely tied to their function.

187. As with the Java API package names, it is my opinion that the Java API class and method names are also short, fragmentary, and descriptive.

188. In the Gingerbread version of the Android packages at issue there are 451 public classes, 133 public abstract classes, and 161 public interfaces. In the Marshmallow version of Android, these data change to 455 public classes, 130 public abstract classes, and 163 public interfaces—again quite similar across different versions of Android.

189. Java API element names, such as class names, are factually descriptive of the underlying functionality, which allows programmers to easily recognize, understand, and use them when reading and writing a program. While this is formally enshrined in the Java Language Specification (discussed above), the rule has more pragmatic roots that date back to the earliest computer languages. The core reason that API component names are short and reflect underlying functionality is that inventive and creative names only loosely tied to the functionality would be difficult for programmers to use.

190. For example, the method "sqrt" is short, simple, and memorable for programmers—and possibly even for non-programmers; a reader of this report may not need to be reminded more than a few times that "sqrt" means "square root." It is technically possible to instead call the square root method "thingthatmultipliedbyitselfyieldstheinput"—that would be more creative, but would inconvenience programmers who had to use the method name. If the authors had wanted to be even more creative, they could have named methods after the

programmers who wrote the implementing code for them, which would result in method names like "Steve." Again, this would be more creative, but it also would be rather inconvenient, because it would make the APIs far less intuitive and therefore harder to use.

191. As an example, it would be possible to build a calculator whose buttons use colors instead of numbers and mathematical symbols. But such a calculator would be difficult to use; a user would have to memorize the colors and their mapping to the underlying numbers and symbols. That would take some time and effort—so much time and effort, in fact, that users are likely to use a traditional calculator instead. Similarly, when the name of a method does not reflect the underlying functionality (as in the case where a square root method is called "Steve" or "thingthatmultipliedbyitselfyieldstheinput") the method would become difficult to learn and use. As a result, in practice all API element names are simple and descriptive.

192. A method whose underlying functionality is to test to see if two values are the same, for example, could be called "equals" or "same," but not much else. Even the longest API element names, such as SQLNonTransientConnectionException, are tied to the underlying functionality. In that case, the name has three parts that demonstrate the underlying functionality. First, the word "SQL" reflects that this relates to the SQL database language (a language that pre-dates Java, and was not created by Sun or Oracle). Second, the word "exception" reflects that this relates to an "exception" (similar to an error message). Both of these terms have been used in software programming for over 30 years, predating Java by some time. A programmer would recognize that the third part of the name ("Non-Transient Connection") reflects underlying SQL functionality that is a term of art from outside the Java language. Because the names of all of these underlying concepts are fixed, or nearly so, the name of the method reflecting these concepts is also necessarily inflexible. The fact that the

name reflects the underlying functionality is not merely convenient—it is practically required to allow the system to be comprehensible to programmers.

193. Class names, like the method names discussed above, are highly functional, in many cases showing only small variations directly related to the class functionality. For example, consider the seven classes whose names end in Event as shown below. These seven classes come from three different packages.

Class Name	**From Package**	**What is the Functionality?**	**Why is it in this Package?**
HandshakeCompletedEvent	javax/net/ssl	A class describing an event that takes place once a Handshake is Completed.	"Handshakes" are part of the Secure Sockets Layer (SSL) networking protocol, and so this is grouped with other "net" and "ssl" features.
PreferenceChangeEvent	java/util/prefs	A class describing an event that takes place when a Preference Changes.	Utilities that user track Preferences must have a way to track what happens when the preferences change, and so this is grouped with "util" "prefs."
NodeChangeEvent	java/util/prefs	A class describing an event that takes place when a Preference Node Changes.	"Nodes" are a common way to organize data. Since these nodes are used to store user preference data, information about the nodes (including changes to them) are grouped with other "prefs"-related functionality.
SSLSessionBindingEvent	javax/net/ssl	A class describing an event that takes place when an	As part of the implementation of the SSL networking

Class Name	From Package	What is the Functionality?	Why is it in this Package?
		SSL Session Binds.	protocol, this is grouped with other "net" and "ssl" classes.
ConnectionEvent	javax/sql	A class describing an event that takes place when a Connection occurs.	Because this is an SQL Connection, it is grouped with other SQL methods.
RowSetEvent	javax/sql	A class describing an event that takes place when an SQL RowSet is changed.	Because this is an SQL Rowset, it is grouped with other SQL methods.
StatementEvent	javax/sql	A class describing an event that takes place when an SQL Statement is changed.	Because this is an SQL Statement, it is grouped with other SQL methods.

194. These names are functional in specifying the purpose of the methods. The noun part of each name that precedes Event describes the event, but there is little creativity in choosing the noun. For example, the HandshakeCompletedEvent describes an event that takes place after the hand-shaking protocol in making an SSL connection has been completed. Similarly, the RowSetEvent class describes an event that takes place when a "rowset" is changed in an SQL database. The names have been chosen not because of deep introspection or creativity on the part of the author, but by simply describing what functionality is contained in the class.

195. Another example of class names that conform to simple rules describing the underlying functionality are the 18 classes whose names end with InputStream and 15 that end with OutputStream. These classes are part of a variety of packages—some are grouped with other input and output functions in java.io and java.nio, but several are part of the packages

java.util, java.util.zip, java.util.jar, java.security and javax.crypto. The table below shows thirteen of the InputStream classes below and their corresponding packages. Again the names for the classes are functional and limited by the responsibilities of each class: the LineNumberInputStream class reads data while keeping track of line numbers while the CipherInputStream uses a cryptographic cipher for reading data.

Class Name	**From Package**	**What is the Functionality?**	**Why is it in this Package?**
FileInputStream	java/io	A stream of inputs from a file.	This is part of the basic input/output ("io") functionality.
PushbackInputStream	java/io	Adds the ability to push data back into an input stream.	This is part of the basic input/output ("io") functionality.
ZipInputStream	java/util/zip	A stream of inputs from a zipped file.	Reading from a zipped file is part of the basic zip file ("zip") functionality.
JarInputStream	java/util/jar	A stream of inputs from a "jar" file.	Reading from a jar file is part of the basic jar file ("jar") functionality.
LineNumberInputStream	java/io	Adds the ability to count the line number to an input stream.	This is part of the basic input/output ("io") functionality.
StringBufferInputStream	java/io	A stream of inputs from a string buffer.	This is part of the basic input/output ("io")

Class Name	From Package	What is the Functionality?	Why is it in this Package?
			functionality.
CipherInputStream	javax/crypto	A stream of inputs that have been passed through a cipher.	Ciphers are part of cryptography, and so functionality to use ciphers is part of the cryptography ("crypto") package.
InflaterInputStream	java/util/zip	A stream of inputs from an "inflater" that "inflates" a zipped file.	Inflaters are part of "zipping" a file and so this is part of util/zip.
FilterInputStream	java/io	Adds the ability to filter to an input stream.	This is part of the basic input/output ("io") functionality.
ObjectInputStream	java/io	A stream of inputs from an object.	This is part of the basic input/output ("io") functionality.
DigestInputStream	java/security	Creates a "messages digest" of a stream of inputs.	Message digests are part of certain security routines, so this class is part of the java/security packages.
ByteArrayInputStream	java/io	A stream of inputs from a byte array.	This is part of the basic input/output ("io") functionality.

196. Java class names are short, fragmentary words and phrases. This is true of other Java API elements, like method names, as well. Most Java API element names are short word phrases, typically of 1-3 words in length. One example is the class "PrintStream," which (not surprisingly) adds printing functionality to output "streams." Elements in the PrintStream class include the method named "append," which appends the argument to the output stream, the method named "print," which prints the stream, and the method named "close," which closes the stream. In fact, every method in the PrintStream class, with only four exceptions, is one word. Two of the exceptions are two words—"setError" and "checkError," which, as one would expect, set and check the error state of the output stream. The other exceptions are "printf" and "println"— abbreviations for "print formatted" and "print line." Besides being brief and fragmentary, these abbreviations have been in use by programming languages since the late 1960s, in ALGOL and C.

197. Other classes, such as the SecurityManager class, have slightly longer names. In this class, three-word method names (such as "checkPackageDefinition") are predominant and there are some four-word method names (such as "checkCreateClassLoader"). But even here, the naming follows a consistent pattern—30 of the 40 methods are named check[SomeProperty], consistently describing their underlying functionality, which is to check the status of the property referred to by the method name. For example, "checkCreateClassLoader" checks to see if it is possible to create a new class loader.

198. Because many classes need the same functionality, and the names of the methods in question are dictated by functionality or by rules (see next section), it is not surprising that many of the names are repeated. The most common names in Oracle's implementation of Java 1.5 are:

Method name	**Number of Times Repeated**	**Functionality?**
toString	194	Converts an object to a String.
equals	157	Tests to see if two objects are equal.
hashCode	147	Creates a "Hash Code" (a numeric representation) of a class.
run	139	Runs the code in the object.
read	96	Reads (typically to a stream of characters).
write	94	Writes (typically to a stream of characters).
remove	88	Removes something (exactly what is removed depends on the class).
get	74	Gets the value of an object.
close	72	Closes a stream.
size	68	Returns the number of items in a collection of items.
clear	61	Clears the content of the thing referenced.
clone	59	Clones the thing referenced.
TOTAL	**1249**	**These 10 method names are used by roughly 1/6 of the methods in Oracle's implementation of Java 1.5.**

199. The organization of the methods into classes, like the organization of classes into a package, is driven by functionality and the requirement that programmers be able to efficiently use these methods. This is why, for example, all of the math functions listed above in Section IV.B.5 are in the same java.lang.Math class.

200. Because these names all describe specific functionalities, limited by design rules to short, fragmentary words and phrases, there is little creativity in the class or method names.

201. The parameter names in the APIs are also functional. Unlike other API elements, the parameter names are not necessarily reused by programmers, who can choose their own names for variables when interacting with a method. But these parameter names still play a functional role because they serve to inform programmers what kind of information the method expects. This functional requirement creates practical restraints on the developer's choice of how to convey information. So, for example, the creators of an API do have the flexibility to call the integer value used by the "abs" function "a," "i," "x," or "Steve." If the value is named "Steve," however, that will still make the documentation and specification of the method unnecessarily confusing to developers who are trying to understand the API.

202. All the parameters in the Java APIs at issue reflect minimal expressive creativity. For example, many methods use parameters that are single letters (such as *a*) that reflect the parameter's roots in algebra. Others are simply abbreviations; for example, at least 41 parameters in Oracle's implementation of Java 1.5 are integers called "i," ("i" being a commonly used abbreviation by programmers for integer variables since long before the Java programming language was created) and at least 23 are characters called "c" (again, "c" being a well-known abbreviation of character). Many others are simple names that reflect the underlying idea being manipulated; *e.g.*, the single parameter name for the constructor JarEntry is named, simply, "name," and the single parameter taken by the method "setSize" is called, appropriately, "size."

5. Many API elements are drawn from the public domain and are not original to Java.

203. Based on my analysis of the names of API elements, it is my opinion that many of them are drawn from the public domain.

204. Java, like many other programming languages, is based on features of previous well-known languages, such as C and C++, including their grammar and syntax. *See, e.g.*,

http://java.sun.com/docs/books/jls/first_edition/html/1.doc.html ("the lexical structure of Java . . . is based on C and C++"); *see also* http://www.gotw.ca/publications/c_family_interview.htm (James Gosling, inventor of Java, quoted as saying "You can go through everything in Java and say 'this came from there, and this came from there'"). Reuse of grammar and syntax from already-familiar languages allowed developers to leverage their existing knowledge and more quickly adopt Java. Similarly, authors of new programming languages often use old method and class names when appropriate, in order to help developers reuse their skills and transition to new languages, and to help make sure the ideas are time-tested. *See, e.g.,* James Gosling's "Feeling of Java" paper, *Computer*, Vol. 30, Issue 6, June 1997, where he writes "Java feels very familiar to many different programmers because Sun had a very strong tendency to prefer things that had been used a lot over things that just sounded like a good idea." As a result, many API element names in modern languages are drawn from the public domain. For example, package names like java.io, java.util, and java.net reflect industry shorthand for common functionality like input/output, utilities, and networking, respectively. These packages with common names then, in turn, contain methods whose naming reflects industry custom and representation of the underlying functionality of the method.

205. In fact, Oracle's own publications acknowledge that Java purposefully derives much of its character from the C and C++ languages, which were well known to computer programmers at the time of Java's development and release. Herbert Schildt, *Java – The Complete Reference* 3-11 (9th ed. 2014).

206. *Java: The Complete Reference* notes that "the similarities with C++ are significant, and if you are a C++ programmer, then you will feel right at home with Java." Furthermore, "if you are an experienced C++ programmer, moving to Java will require very little

effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java." *Id* at 11.

207. Among the similarities between Java, on the one hand, and C and C++, on the other, *Java: The Complete Reference* explains as follows:

> Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object oriented features were influenced by C++.

Id at 3.

> Java derives much of its character from C and C++. This is by intent. The Java designers knew that using the familiar syntax of C and echoing the object-oriented features of C++ would make their language appealing to the legions of experienced C/C++ programmers.
>
> In addition to the surface similarities, Java shares some of the other attributes that helped make C and C++ successful. First, Java was designed, tested, and refined by real, working programmers. It is a language grounded in the needs and experiences of the people who devised it. Thus, Java is a programmer's language. Second, Java is cohesive and logically consistent. Third, except for those constraints imposed by the Internet environment, Java gives you, the programmer, full control.

Id at 7.

208. Other publications also acknowledge the similarities between Java, C, and C++. For example, Jeff Friesen observes that "Java's *syntax* (rules for combining symbols into language features) is partly patterned after the C and C++ languages to shorten the learning curve for C/C++ developers." Jeff Friesen, *Learn Java for Android Development* at 2 (2nd ed. 2013). More specifically, "[m]any of Java's reserved words are identical to their C/C++ counterparts (for, if, switch, and while are examples) and C++ counterparts (catch, class, public, and try are examples)." *Id.* Java also "supports character, double precision floating-point,

floating-point, integer, long integer, and short integer primitive types, and via the same char, double, float, int, long, and short reserved words." *Id.* at 3.

209. Some examples of functions that are very similar in Java and the pre-existing C and C++ languages as a result of their functionality and industry custom are shown below:

Name	**Originated in?**	**Java equivalent**	**What is it?**
Char	At least C; *see* C Reference Manual, Dennis Ritchie, 1975 (*available at* http://www.cs.bell-labs.com/usr/dmr/cman.ps)	Char	A data type holding a character; used repeatedly in method names, such as C++ "getchar" and Java "getChars." Other data types, such as *int* and *double*, also date back to C and at least the 1970s.
int abs (int i)	At least C; *see* C Reference Manual.	public static int abs (int a)	A function, returning the absolute value of the argument.
printf()	At least C; *see* C Reference Manual.	printf() (part of the java.io package)	A function, printing a formatted string to the screen or other output device.

210. Indeed, many of the names and concepts in Java have been used by the industry for decades. For example, the C Reference Manual references "int," "double," and "char," all used in Java. The "bool" data type, which became "boolean" in Java, dates back to at least ALGOL in 1968, and is a direct reference to boolean logic—invented in the 1800s.

211. Another example of this is the java.util.regex API package, which implements "regular expressions"—a standardized way of testing if a given string of characters matches a particular pattern. Regular expressions were first formalized in 1968 ("Programming Techniques: Regular expression search algorithm," Ken Thompson, Communications of the

ACM, Vol. 11, Issue 6, June 1968) and were known by the abbreviated name used by Java (regex) at least as early as 1983. Java.util.regex has two classes: Pattern, and Matcher. Method names in the Pattern class are compile, flags, matcher, matches, pattern, and split, while method names in the Matcher class include matches, pattern, reset, and start. Each of these names—particularly the extremely common "pattern" and "matcher"—are used in publicly available regular expressions software that predate java.util.regex, and all of them are discussed in *Mastering Regular Expressions*, Jeffrey E. F. Friedl, O'Reilly and Associates, 1997, which Oracle's documentation for java.util.regex cites. Sun also used third-party source code (Jakarta Regexp) to implement java.util.regex, and this code also had references to many of these terms, including "compile," "pattern," and "match."

212. During the prior trial in this case, I was present in the courtroom when Dr. Joshua Bloch testified that Sun had "reimplemented the Perl regular expression API in Java." RT 803:21-804:21. In other words, Sun had taken the manner in which regular expressions were implemented in the Perl language and re-implemented those exact expressions as part of the java.util.regex API in Java. As with Google's use of the 37 API packages in Android, Sun "got the API from somewhere else, but we did our own independent implementation." *Id.*

213. Similarly, the java.sql and javax.sql packages are also based in part on pre-existing terms widely used in the industry. The package names themselves are a reference to the SQL standard, originally introduced in the academic literature as SEQUEL in 1974 (SEQUEL: A structured English query language, Proc. ACM SIGFIDET Workshop, May 1974, pp. 249-264). The classes and methods frequently are named after SQL concepts, and in particular (according to Sun's documentation at http://jcp.org/aboutJava/communityprocess/first/jsr054/jdbc-3_0-pfd-spec.pdf) on the X/Open SQL Call Level Interface (CLI), which dates to the first half of the

1990s (*available at* http://pubs.opengroup.org/onlinepubs/009654899/toc.pdf). For example, java.sql includes classes named "Array," "Blob," "Clob," and "Ref," which are the names of data types from the SQL standard. Similarly, the SQL CLI standard defines a method called "prepare" that operates on a StatementText. Java.sql's Connection class has a method called prepareStatement that has similar functionality. The SQL CLI standard also uses "commit" and "rollback" to discuss specific actions that can be done to a database, and java.sql's Connection class has matching "commit" and "rollback" methods that perform the actions discussed in the standard.

214. Java.util.zip is another example where the name of the package, and at least some API element names within the package, are references to terms that substantially predate Java's use of the terms. In this case, "zip" is a reference to the zip file format that has been in use since before the creation of Java. Class names in this package include "Adler32" (named after the Adler-32 algorithm invented by Mark Adler and licensed to the public as part of the zlib library) and "CRC32" (named after the CRC-32 algorithm, which dates back to the 1970s). Within the java.util.zip classes, method names include "deflate," "inflate," and "setDictionary," which are very similar both to general industry terms for these processes but also to the specific function names "deflate," "inflate," and "deflatesetdictionary" that are in the publicly available open source library (zlib) that predates, and is incorporated by, Java. (*See* http://www.zlib.net/manual.html).

215. Java has likewise borrowed many of its methods from math.h, which is the header file in the standard library of the C programming language for basic math operations. Some

sample methods in which Java used the method signature as it existed in C, only inserting the word "static" at the beginning, follow.[15]

Math.h	**Java**
double atan2 (double y, double x)	static double atan2 (double y, double x)
double expm1 (double x)	static double expm1 (double x)
double cosh (double x)	static double cosh (double x)
double hypot (double x, double y)	static double hypot (double x, double y)
double log1p (double x)	static double log1p (double x)
double sinh (double x)	static double sinh (double x)
double tanh (double x)	static double tanh (double x)
double ceil (double arg)	static double ceil (double a)
double cos (double arg)	static double cos (double a)
double floor (double arg)	static double floor (double a)
double log10 (double arg)	static double log10 (double a)
double pow (double base, double exponent)	static double pow (double a, double b)

216. In other packages, Java has taken the names of certain constants wholesale. For example, in the java.net.SocketOptions interface, each of the following constants appear to have been taken directly from the POSIX standard: IP_MULTICASE_IF; IP_MULTICASE_LOOP; IP_TOS; SO_BROADCAST; SO_KEEPALIVE; SO_LINGER; SO_OOBINLINE; SO_RCVBUF; SO_REUSEADDR; and SO_SNDBUF. Given the generally verbose naming conventions used in Java, it is hard to draw any conclusion other than that these terse constant names were copied from the POSIX standard.

[15] Note that these include examples from math.h in which Java did no more than make minor changes of the variables to be used by the function, for example by changing the variable from "arg" to "a."

217. Further copying from POSIX is apparent in the methods made available for the java.net.SocketImpl class. The names of five of the main methods in that class—accept, bind, connect, listen, and close—are precisely the same as what is used in the POSIX API for managing sockets.

218. Similarly, the inputs to java.util.Formatter are taken directly from those used by the printf() function of the POSIX standard. Thus, in both systems "%c" refers to a character; "%s" refers to a string; "%d" refers to a decimal number; "%o" refers to an octal number; "%e" refers to a number in scientific notation; "%f" refers to a floating point number; "%a" refers to a hexadecimal floating point number; and "%%" refers to a literal percent sign.

219. Nor is the manner in which, or names by which, Java organizes its classes novel. A comparison to the Smalltalk-80 language, which was created long before Java, shows a striking similarity between the classes used there and the classes found in Java:

Smalltalk-80	**Java**	**Underlying concept**
Object	java.lang.Object	The base class from which all other classes inherit
Character	java.lang.Character	A single character in a string
Date	java.util.Date	A specific instant in time
Time	java.sql.Time	Utilities for dealing with time
Number	java.lang.Number	A superclass for numbers
Float	java.lang.Float	An IEEE-754 floating-point number
Integer	java.lang.Integer	An integer number
Collection	java.util.Collection	A superclass/interface for data structures organizing groups of objects
LinkedList	java.util.LinkedList	A sequence of nodes, each containing a value

Smalltalk-80	Java	Underlying concept
		and a pointer
Semaphore	java.util.concurrent.Semaphore	A concurrency primitive useful for enforcing mutual exclusion
Array	java.lang.reflect.Array	A sequence of indexed elements
String	java.lang.String	A sequence of characters; text
Set	java.util.Set	A collection of elements containing no duplicates
Dictionary	java.util.Dictionary	A collection of key-value pairs indexed by key
Stream	java.util.stream.Stream	A sequential view of an underlying resource
Random	java.util.Random	Utilities for generating pseudorandom values
Boolean	java.lang.Boolean	True or False

220. This shows that even the broad categories into which Java places its various classes and methods were incorporated from pre-existing languages.

C. The Amount and Substantiality of the Portion Taken

221. As demonstrated above in the example of Math.sqrt, using the name is necessary to invoke the underlying functionality. It is also the *only* way to invoke the underlying functionality—one cannot, for example, change "sqrt" to "square_root" in the example above and still expect the code to work. Nor could one change the number or type of arguments. Programs, unlike the human operators of our calculator and car analogies, are not flexible—they must be fed precise information in order to operate.

222. In using the method declarations of the 37 API packages, Google used only a small portion of Oracle's copyrighted work. This is true regardless what one considers the relevant work for purposes of this comparison. If one compares the method declarations to the entire Java SE platform, the amount is miniscule; Android does not use any of the Java runtime,

or virtual machine. Furthermore, Oracle's copyright claim is based on only 37 of 162 API packages that comprise Java SE. Finally, even within those 37 API packages, Google used only the method declarations, and not the much larger base of implementing code.

223. I focus my discussion below on the method declarations within the context of the 37 API packages, though my opinions regarding the amount and substantiality of the portion taken would apply with even more force if the method declarations are compared to the 162 API packages or to the Java SE platform as a whole.

224. Of the 37 API packages, Google used only the package, class, and method names; besides the method declaration, Google implemented all of the code independently. Oracle's allegations suggest that the method declarations amount to about 7000 lines of code, which—based on the calculations above—amounts to only about 2.6% of the implementations of the 37 API packages at issue, 0.4% of the overall Android (Gingerbread) Runtime Core Libraries, and only 0.04% of the size of the entire Android source code.

225. As discussed above, every API, including the Java APIs at issue in this case, exists in two forms: the method declaration of the API (comprised of those elements—name, arguments, and return) and the implementation of the API. The implementation is the actual underlying source code that implements the API and allows the API to function. Any two implementations of the same API will have some portions that are the same, because each implementation must include exactly the same method declaration, including all the elements of the declaration, such as the arguments and return values, in order to be compatible. However, the overall source code may—and indeed does—differ significantly from implementation to implementation.

226. Even if only a small fraction of the source code of two implementations is identical, the remaining code may appear similar to the untrained eye, both because certain key lines (the method, package, and class declarations) must be the same, and because practical considerations will constrain the expression of the code implementing the functionality. For example, there may be both efficient and inefficient ways to implement a given method, but programmers will typically choose the most efficient way. Similarly, coding standards relating to indentation, punctuation, and the like will also constrain how code is written.

227. In addition, many programmers have learned by studying and reading source code, so they typically write code in a style similar to others. Returning to the car analogy, there may be unusual ways to power a car (hydrogen, rotary engines, etc.), but in most cases the solutions will end up looking similar to other implementations for practical reasons and standard design practices, and not because the car manufacturers were copying from each other.

228. An API implementation that uses only the necessary API components, but does not repeat the underlying implementation, is an "independent" implementation. A Ford and a Chevy are, in this sense, independent implementations of a car—while they both provide drivers with a gas pedal and steering interface to the underlying functionality, Chevy engineers likely did not copy Ford's plans to build the Chevy's engine and steering mechanism. Similarly, the fact that virtually every modern computer application supports common keyboard commands like Ctrl+C, Ctrl+V, and Ctrl+P does not prove that the programmers used each other's implementation source code. Instead, they have each re-implemented the functionality in a way that makes sense for their circumstances, reusing only the "interface" of the commonly known keyboard commands.

229. I illustrate below how Android's implementations differ from Oracle's implementations.

BEGIN ORACLE SOURCE CODE

230. The first example is one I have used earlier: the Math.abs function. As discussed above, the absolute value of an integer is essentially the magnitude of the integer, *i.e.*, the distance of the integer from zero. As also discussed above, the *declaration* of the method (the function name, return type, and parameter type) is specified as part of the Math.abs API and must be the same in any implementation of the Math.abs API in order to provide compatibility. Again, the following chart shows the various identical method declarations for abs from different implementations of the Java APIs:

Java SE:	public static int abs(int a)
Harmony:	public static int abs(int i)
GNU Classpath:	public static int abs(int i)
Android:	public static int abs(int i)

(As I explain elsewhere, the variable name chosen for the parameter in the parentheses need not be the same, and, in fact, the variable name in the Android implementation is different than in Oracle's implementation.)

231. Because the concept is so simple ("if the number is negative, give the positive version of it") the implementations are short—all it takes is one line for the declaration, and one line for the actual functionality. Despite this simplicity and brevity, Oracle and Android's implementations are different. The table below shows the Android source code that implements the Math.abs function in the java.lang.Math class compared to the source code that implements JDK1.5 code.

Android Math.abs	Oracle JDK 1.5 Math.abs

`public static int abs(int i) {`
`  return i >= 0 ? i : -i;`
`}`
	`public static int abs(int a) {`
	`  return (a < 0) ? -a : a;`
	`}`

232. As required by the API, the first line of the method—the function name, return type, and parameter type—are essentially identical in both implementations. The name of the parameter—*a* for the JDK1.5 implementation and *i* in the Android implementation—is the only thing different. The parameter name can be different because the name of the parameter is not part of the API. The parameter type, *int*, on the other hand, must be the same if the two implementations are to be compatible.

233. The actual implementation of the method—the second line, shown in blue—is how the absolute value is calculated. Each of these lines of code is different, but correct. Put into English, the line of code from the Android implementation translates to "if the parameter *i* is greater than or equal to zero, return *i*, otherwise return *i*'s negation." In the JDK1.5 implementation the code translates to English as "if the parameter *a* is less than zero, return *a*'s negation, otherwise return *a*." While these implementations must capture the same functionality, and bear some similarity because of the requirement that the method name and arguments be the same, they capture the functionality with different implementations.

234. The second example illustrates how the functionality expressed by an API is implemented differently in the java.lang.String class method String.compareTo. In programming, a "string" is a sequence of characters, such as a word or sentence. The compareTo method compares two strings, in order to determine whether one string is less than, equal to, or greater than another string. In programming, a string that is "less than" another

string is alphabetized first. For example, if "compareTo" was used to compare "apple" to "cat," the method would indicate that "apple" is less than "cat."

235. Below is the Android and Oracle JDK 1.5 source code that implements the compareTo method.

#	Android String.compareTo	Oracle JDK 1.5 String.compareTo
1	`public int compareTo(String string) {`	`public int compareTo(String anotherString) {`
2	`  // Code adapted from K&R, pg 101`	`  int len1 = count;`
3	`  int o1 = offset, o2 = string.offset, result;`	`  int len2 = anotherString.count;`
4	`  int end = offset + (count < string.count ? count : string.count);`	`  int n = Math.min(len1, len2);`
5	`  char[] target = string.value;`	`  char v1[] = value;`
6	`  while (o1 < end) {`	`  char v2[] = anotherString.value;`
7	`    if ((result = value[o1++] - target[o2++]) != 0) {`	`  int i = offset;`
8	`      return result;`	`  int j = anotherString.offset;`
9	`    }`	
10	`  }`	`  if (i == j) {`
11	`  return count - string.count;`	`    int k = i;`
12	`}`	`    int lim = n + i;`
13		`    while (k < lim) {`
14		`      char c1 = v1[k];`
15		`      char c2 = v2[k];`
16		`      if (c1 != c2) {`
17		`        return c1 - c2;`
18		`      }`
19		`      k++;`
20		`    }`
21		`  } else {`
22		`    while (n-- != 0) {`
23		`      char c1 = v1[i++];`
24		`      char c2 = v2[j++];`
25		`      if (c1 != c2) {`
26		`        return c1 - c2;`
27		`      }`
28		`    }`
29		`  }`
30		`  return len1 - len2;`
31		`}`

236. As noted in a comment on line 2 of the Android implementation (on the left), the Android implementation of compareTo is adapted and based on code from "K&R," a reference to "The C Programming Language," a book written by the C language's principal authors, Brian Kernighan and Dennis Ritchie. The K&R book, and the code contained within it, were published

long before the Java language existed. The body of the function—that is the code between and including the function's curly braces (*i.e.*, the "{" and "}" that mark the beginning and end of the source code for a function)—is 11 lines long.

237. In the Oracle JDK 1.5 implementation on the right, the first part of the first line of the implementation is the same as the Android implementation on the left—"`public int compareTo(String`". Again, this similarity is required for compatibility. Use of the same parameter name, however, is not required for compatibility, and so the parameter named *string* in the Android implementation is instead *anotherString* in the Oracle JDK 1.5 implementation. The Oracle implementation is also 31 lines, instead of the Android implementation's 15, indicating again that different algorithms and language features were used to reach the same result.

238. These two implementations are functionally identical—they compare the corresponding characters of two strings—but the actual code is very different. For example, in comparing the string "catastrophe" to "catalog" the code scans the first four characters, and finds that they are the same. It then determines the relative order of the strings by comparing the fifth characters—s in catastrophe and l in catalog. In the Android implementation the two characters compared are captured by the expressions `value[o1++]` and `target[o2++]` whereas in the JDK1.5 implementation these characters are stored in variables `c1` and `c2` and are captured by the expressions `v1[i++]` and `v2[j++]` in one part of the code and `v1[k]` and `v2[k]` in a different part of the code. In both versions of the code, once a difference in characters is detected (*i.e.*, s and l in the catastrophe and catalog example), the code need not compare further characters to determine the relative order of the strings. For example, in comparing "ant" and "bee" comparisons stop after the first characters have been examined, but when comparing "distance" and "distant" the function can only determine the relative order after examining the

seventh character of each string (c and t). Despite the similar functionality, the code that performs these comparisons and looks at the corresponding characters of each string is very different.

239. To further illustrate how the same compareTo API can be implemented in various ways, the GNU Classpath implementation of the String.compareTo method is shown in the following table, and is different from both the Android and Oracle JDK 1.5 implementations. Again, all of these sets of source code implement the same underlying functionality—they compare two strings of characters by examining each individual character until corresponding characters are different. The method name, return type, and parameter type ("`public int compareTo(String`") are again identical, as they must be for compatibility and interoperability. However, the way these sets of source code actually achieve this functionality differs significantly. For example, the Android implementation uses variable names `o1` and `o2` whereas the Classpath implementation uses variables `x` and `y`. The Android and Classpath implementations (unlike the Oracle implementation) both use a concept called a "while" loop that repeats a given operation "while" a particular condition is true, but the loop in the Android implementation uses the condition `while (o1 < end)` whereas the loop in the Classpath implementation uses the condition `while (--i > 0)`. And again, like the Android and Oracle implementations, these implementations are of different length, though the difference is much smaller. Although the logic used in the Android and Classpath implementations is the same, the implementations are very different.

#	**Android String.compareTo**	**GNU Classpath String.compareTo**
1	`public int compareTo(String string) {`	`public int compareTo(String anotherString)`
2	`  // Code adapted from K&R, pg 101`	`  {`
3	`    int o1 = offset, o2 = string.offset, result;`	`    int i = Math.min(count, anotherString.count);`

```
4       int end = offset + (count < string.count ?
        count : string.count);
5       char[] target = string.value;
6       while (o1 < end) {
7         if ((result = value[o1++] - target[o2++])
          != 0) {
8           return result;
9         }
10      }
11      return count - string.count;
12  }
13
```

```
    int x = offset;

    int y = anotherString.offset;
    while (--i >= 0)
      {

        int result = value[x++] -
        anotherString.value[y++];
        if (result != 0)
            return result;
      }
    return count - anotherString.count;
  }
```

240. The final example involves comparing implementations of the class ZipFile from the package java.util.zip. This class manipulates "zip" files, which are files that contain one or more other files, so that those files can be easily emailed, stored, and otherwise moved around. Because zip files are archival, they allow many files or folders to be packaged together as a single zip file. In addition, zip files are "compressed"—that is to say, a zip file is usually smaller than the sum of the sizes of the files contained in the zip file. Each of the files stored in a zip file is referred to as an "entry" in the zip file.

241. The Java API package java.util.zip contains several classes for creating, reading, writing, and manipulating zip files and the files ("entries") stored within them. In particular, I will focus on the class ZipFile and the method getInputStream from that class in order to compare and contrast an API with its implementation.

242. Among the public methods in ZipFile is one called getInputStream, which is used to "read" a zip file—*i.e.*, to access the archived and compressed contents stored in a given zip file. The getInputStream method does this by creating an "InputStream," which is a standard way for Java programmers to access files and other data sources. An InputStream is essentially a representation of a steady stream of information. Programs written in the Java language can act on these streams in a variety of ways, such as reading the next piece of data in the stream, skipping ahead to another part of the stream, and finding out how much of the stream is still

available to be read. When a program written in the Java language opens, closes, and read documents or other files, the program is using an input stream.

243. This functionality—both the ZipFile class generally and the getInputStream method specifically—can be implemented in a variety of ways. As I will discuss in more detail below the implementation of a class can contain both "public" methods—or methods that can be used by any programmer when writing programs—and "private" methods—or methods that can only be used by the code implementing the class, and used only for the purpose of implementing other parts of the class. "Public" and "private" methods can also be thought of as "external" and "internal" methods, respectively—public methods can be used from outside of the program, while private methods are "internal" to the program and can only be used by that program, not by other programs. For one class to be compatible and interoperable with another class, both must have the same public methods, but they may have different private methods and still be compatible. The Android implementation of ZipFile contains two private methods used to help implement the public methods. The Oracle JDK1.5 implementation of ZipFile, in contrast, contains 20 private methods. The GNU Classpath ZipFile.java implementation contains seven private methods. This significant difference in the number of private methods illustrates that although the public methods of the API are similar, as they must be, the internal implementations of these methods and the class ZipFile are very different.

244. Just as the ZipFile classes in these two implementations as a whole are different, the getInputStream method in each is also different. Both the Oracle and Android implementations of the getInputStream method accomplish the same task: when given a "ZipEntry" object (*i.e.*, a reference to one of the files or directories in a zip file), return an input stream that allows the program to read that entry. However, the source code that implements

Oracle Java 1.5 method ZipFile.getInputStream, including the private helper methods and classes it uses, is 275 lines of code. Android's implementation of the same method, including its private classes and methods, is 120 lines of code. (Because of their length, the table with this code is attached as Exhibit F.) This is a very large difference in how the methods are implemented.

245. However, it is not just the length of the two implementations that distinguish them. They are also structurally different, which can be seen by analyzing the "private" methods and classes used in the implementations. Both the Android and JDK 1.5 methods use private classes to represent the input stream that corresponds to the file or directory being read. Android's implementation uses two internal classes, named RAFstream and ZipInflaterInputStream.[16] These classes "extend" (*i.e.*, are based on and add new functionality to) other classes—InputStream and InflaterInputStream, respectively. The Oracle JDK 1.5 implementation of ZipFile.getInputStream also uses two internal classes, but they have different names from Android's internal classes—ZipInputStream and MappedZipInputStream. Besides the different names, the second of the two classes (MappedZipInputStream) is based on ZipInputStream, rather than InflaterInputStream (as in the Android version). In the JDK 1.5 code there are three private methods (highlighted in the table below in blue) whereas there are none in the Android implementation. Again, the usage of structurally different private methods and classes indicates, in my opinion, that the implementation of these specific methods are very different, and more generally, shows how analysis of private methods can be used to help understand whether or not two given implementations are similar.

246. The methods in the source code that implements the complex task of creating the InputStream differs, but that is not the only difference—a more detailed analysis shows that even

[16] Technically these are not private - they can be used by other parts of the API package. However, the classes are only used within the ZipFile.java file, and can't be used by external programs, so they are effectively private.

the relatively simple programming task of ensuring that the ZipFile has a name is implemented differently. The fragment of the ZipFile.getInputStream source code that implements this simple functionality is shown in the table below. The Oracle JDK 1.5 implementation begins this task by calling a private method that is not used anywhere else in the package. In turn, this method calls an error message (called an "exception") if the ZipEntry is "null"; *i.e.*, if the ZipEntry has no name.

247. The Android version has several key differences. First, it does not use a helper function—it does the work itself. Second, if the FileEntry has no name, the Android code simply returns "null"—*i.e.*, an empty value—instead of a formal error code (the "exception" mentioned above). Third, the Android source code finds the name of the Entry in a different way from the Oracle code—the Oracle code directly accesses the value of the name, using the statement entry.name, rather than using what is known as an "accessor function"—represented in the Android code by entry.getName(). While this difference may look subtle, the approach used by the Oracle code is generally considered bad style; when an accessor function is provided (as is the case here), it should be used instead of directly accessing the information.

#	**Android ZipFile.getInputStream [fragment]**	**Oracle Java 1.5 ZipFile.getInputStream [fragment]**
1	`public InputStream getInputStream(ZipEntry entry) throws IOException {`	`public InputStream getInputStream(ZipEntry entry) throws IOException {`
2	`  entry = getEntry(entry.getName());`	`  return getInputStream(entry.name);`
3	`  if (entry == null) {`	`  }`
4	`    return null;`	`private InputStream getInputStream(String name) throws IOException {`
5	`  }`	`  if (name == null) {`
6	`...`	`    throw new NullPointerException("name");`
7		`  }`
8		`  long jzentry = 0;`
9		`  ZipFileInputStream in = null;`
10		`  synchronized (this) {`
11		`    ensureOpen();`
12		`    jzentry = getEntry(jzfile, name, false);`
13		`    if (jzentry == 0) {`

```
14                                          return null;
15                                        }
16                                        if (mappedBuffer != null) {
17                                          in = new
                                            MappedZipFileInputStream(jzentry,
                                            name);
18                                          } else {
19                                            in = new
                                              ZipFileInputStream(jzentry);
20                                          }
21                                        }
22  ...
```

248. By looking closely at ZipFile.getInputStream, I have shown that the same, compatible, interoperable functionality can differ in many ways—overall, by simply comparing the length of the two implementations; at an intermediate level, by showing that there are different names and numbers of private methods and classes used to implement the functionality; and at a granular level, by showing that one particular subtask is implemented in different ways.

END ORACLE SOURCE CODE

249. In each of these three methods examined in this section, I have shown that the programmatic logic used to implement a particular method can be very different, with only one small portion—the method name and argument types—being the same. These files are typical of all Android and Oracle JDK 1.5 files that I have inspected—one small portion, which is required to be the same for purposes of compatibility and interoperability, is the same, and the rest of the file is different.

250. As discussed above, the names and parameters of the APIs must be the same for many reasons. While the Android platform is compatible with and provided the functionality of the Java language APIs at issue, and necessarily uses the same API names and organization in order to do so, my opinion, after my review of the Android and Oracle source code, is that Android's underlying implementation (or source code) of the APIs is substantially different from Oracle's implementation. That is, besides the overlapping use of the same method declarations,

Android's implementing code for the 37 API packages is substantively different. In my opinion, this demonstrates that Google used only the bare minimum of code from the 37 Java API packages in a way that comports with the free and open nature of the Java programming language.

251. Besides the kind of line-by-line analysis above, we can analyze the differences in the implementations of the APIs by examining the names of the private methods of each implementation. In my opinion, the different names for these private methods generally demonstrate how different the implementations of the Android source code are from the implementations found in the Oracle JDK 1.5 source code.

252. As touched on above, "public methods" are the methods that are made available for use by programmers who use an API to write applications. These must be the same if the two implementations are to be compatible. In contrast, "private methods" help to implement the API but are not visible or available for use by software developers building their own software. The classes that are at issue in this case have public methods that must be implemented in order to be compatible with the API, *e.g.*, Math.abs and Math.sqrt in the java.lang package. However, the API does not dictate how the methods are implemented. I demonstrated above in my analysis of getInputStream that private, helper functions are often used in implementing the public methods required by the APIs. Differences in the private methods typically reflect differences in the implementations (while similarities may be due to various reasons, including using otherwise common naming conventions and terminology). For example, one way to evaluate the differences in the implementations above is to list the names of the private methods, and compare the two. To the extent the names and quantity of the private methods in the two implementations are different, that suggests that the implementations themselves are also

different (again, there may at times be overlap when programmers use common terms and naming conventions, similar to re-using the names of functions like "sqrt" and "abs"). For example, the getInputStream method is implemented using different private methods and private classes in the different implementations—the Android implementation uses three private methods and two private classes, whereas the Oracle JDK 1.5 implementation uses two private classes but no private methods. This difference in the number of the private methods and classes (and in many places, also the type and name of the internal structures) indicates that the two implementations have very different underlying structures and therefore are not similar.

253. Using software I developed to analyze the classes examined in this report, I detected large differences in how public and private methods are used across the Android, GNU Classpath, and Oracle JDK 1.5 implementations. I used the program (attached as Exhibit G) to examine the accused packages, and created the table below to summarize the data for the 740 public classes and interfaces in common between the Android and Java implementations of the 37 accused packages. For comparison, I have also provided information on the GNU Classpath implementation of the same materials.

254. The column labeled "Total Methods" provides the total number of methods (including constructors) found across all classes. The column labeled "Total Private Methods" shows how many of these methods are labeled as private, and hence not accessible to programmers but used to implement the public methods. As I discussed in the example of the getInputStream method in the java.util.zip class, sometimes private methods are used to implement the public methods, but they are not part of a class's API because programmers using the class cannot access the private methods. The column labeled "Percent Private" provides one estimate of how often private methods are used across all classes. Each of these classes

contributes a percentage between zero and one hundred to a running total. If all methods in a class are private, the percent private for that class is 100%. If all methods are public and none are private, the percent private is 0%. The percentage shown in the column is the average of these per-class percentages across all classes. This metric, and the significant differences between the Android and Oracle implementations, shows that the Android classes use, on average, fewer private methods than the other Java implementations—which, in my opinion, indicates that the implementations are significantly structurally different. The structural differences between the implementations are also indicated by the total number of methods that differ across the implementations. Methods can be public, private, or package access, and it is possible to add public methods that are not part of the API. The differences between the total number of methods across the implementations is a further indication that the implementations of the APIs are very different.

	Packages	Total Methods	Total Private Methods	Percent Private
Android	37	8994	970	14.2%
GNU Classpath	37	7365	576	24.53%
JDK1.5	37	8190	1369	22.3%

255. Again, the substantially different numbers of classes and methods, and the different ratio of public to private methods, demonstrates that each of the implementations measured is substantially different from the other. In particular, recall that private methods, unlike public methods, are not required to be the same. As a result, the very different number of total private methods in the implementations of the allegedly infringed packages leads me to conclude that, when the authors of the three pieces of software were not constrained by compatibility, they took very different routes to implement the functionality. My direct

inspection of a cross-section of the files at issue confirms the results of this numerical approach. As expected from a review of the overall numbers, in the individual classes, the number of private methods and classes, and their underlying implementation, also vary substantially between the two implementations.

D. The Effect of the Use Upon the Potential Market

256. In my opinion, Android is not a substitute for Java SE because that Java platform targets a substantially different market.

257. To the extent that mobile device manufacturers previously licensed Java SE and subsequently chose to use Android instead, I believe that is not because Android merely supplants Java SE—it is because Android is a different platform that is far more technically suited for mobile devices than Java SE. In fact, the example of Android well illustrates the technical flaws inherent in Java SE for mobile devices; as noted above in ¶ 152, Sun and Oracle's efforts to create a full stack smartphone using Java SE all failed.

258. Even if, counterfactually, Android could have had some effect on the potential market for Java, that effect would have come about as a result of Sun/Oracle's own actions. In 2007, Sun chose to make Java SE open-source software by releasing the OpenJDK under the GPL-2.0-CE license. This meant that anyone who wished could use the source code for the Java class libraries, including the libraries represented by the 37 API packages at issue in this case, free of charge so long as they complied with the terms of the GPL-2.0-CE license. It also meant that anyone who wanted to simply use/license the API method declarations (and either modify or throw out the existing OpenJDK implementing code) could have done so freely, provided compliance with the GPL-2.0-CE license. *See, e.g.*, Smith Dep. Tr. at 224:13-225:11. The GPL-2.0-CE license does not require compatibility with Sun/Oracle's implementation or the TCK. *Id.* at 226:1-5.

259. In its related Java FAQ, Sun acknowledged the possible reuse of only certain packages included under the GPL-2.0-CE license. Specifically, Sun recognized that, by releasing OpenJDK under the GPL-2.0 and GPL-2.0-CE licenses, it was allowing for the creation of Java implementations that might not comply with the Java specification:

> Q: Can someone create and distribute an implementation that isn't compatible with the Java specification using this code?
>
> A: Yes. We do not recommend or endorse that action, however. In addition, they cannot label that implementation with the Java Compatible or Java Powered (for Java ME) brand and logo. These brands are your assurance that an implementation has passed the relevant TCKs.
>
> Q: Can someone create software that doesn't even implement Java, but uses pieces of the OpenJDK code commons? What are the limitations, if any?
>
> A: Yes. There are no limitations. But there is an obligation to meet the requirements of the GPL (plus Classpath and Assembly exceptions if appropriate).

GOOG-00000221 (at -240); GOOG-00000316 (at -335).

260. Sun similarly acknowledged that open-sourcing the Java Platform (under the GPL-2.0 license and GPL-2.0-CE license) might lead to implementations on platforms not supported by Sun:

> Q: What impact do you see open sourcing the JDK will have on bringing Java to platforms that Sun may not be directly interested in supporting? For instance, could the community provide an implementation for a game console? Can a bunch of like-minded engineers simply approach the console manufacturer and offer to compile a JVM for them for free from the OpenJDK sources?
>
> A: Great question - because we think one of the most important benefits of open-sourcing Sun's platform implementations will be the new platforms that will be supported by the community. And to your point - once the complete JDK is available under the GPL, it will be free software, and developers will be able to use it in any way they see fit, as long as they abide by the GPL, including publishing any modifications to

> the platform they might make, benefiting the whole Java technology ecosystem. We think there are many opportunities to bring Java to new platforms, and neither licensing nor technology really stands in the way in most cases.

GOOG-00000221 (at -253); GOOG-00000316 (at -348).

261. The OpenJDK project includes releases for Java SE 6 and later versions of Java SE. *See* http://mail.openjdk.java.net/pipermail/jdk6-dev/2008-February/000001.html. In the analysis below I used this Java SE 6 version. `hg clone` http://hg.openjdk.java.net/jdk6/jdk6.

262. I created a script to compare the source code for Java SE 1.5/5 with this earliest version of the OpenJDK project. Every one of the 37 API packages in the Java SE 1.5 code is included in the OpenJDK project. Additionally, two files in the Java SE 1.5 java.nio and java.nio.charset packages that are templates, not code per se, but files used to generate code, are also present as template files in the OpenJDK release. The template file java.nio.X-Buffer.java in the Java SE 1.5 codebase is present as java.nio.X-Buffer.java.template in the OpenJDK 6 code base. These template files are used to generate java.nio classes FloatBuffer, ByteBuffer, IntBuffer, DoubleBuffer, LongBuffer, ShortBuffer, and CharBuffer for example.

263. Every public method in the Java SE 1.5/5 codebase is also present in the OpenJDK codebase. When the script identified a method in the Java SE 1.5/5 codebase as not present in the OpenJDK codebase, examination of each class and each method showed that the methods were, in fact, present in the OpenJDK codebase and were semantically identical even when the script showed syntactic differences. The script identified roughly 65 such examples, and I found each example of a public method present in the OpenJDK codebase. Examples included methods in public interfaces that did not include 'public' in the OpenJDK equivalent interface (the Java Language Specification indicates that all methods in a public interface are

public); default or parameterless constructors with no method body included in the Java SE 1.5 codebase for abstract classes, whose implementation is equivalent to not including the constructor (in many cases these constructors were in classes in the javax.crypto classes where the author of the class identified in the Java SE codebase was the same author in the OpenJDK codebase); and methods identified as final public rather than public final where both formats are equivalent.

264. I understand that Google is now basing its implementation of the 37 API packages on OpenJDK, taking this code under the GPL-2.0-CE license under which Sun/Oracle publicly released OpenJDK. Although Google did not take the OpenJDK license until 2015, there is no technical reason it could not have done so in the 2007/2008 timeframe as well. Indeed, as discussed above, Sun/Oracle's own FAQ page for OpenJDK during this timeframe acknowledged that "[t]here are no limitations" on the use of source code found in OpenJDK other than those set forth the terms of the OpenJDK license. GOOG-00000221 (at -240); GOOG-00000316 (at -335); GOOG-00597967 (at -978).

265. Thus, Sun/Oracle's own actions ensured that any person or entity who chose to do so could make use of the Java SE APIs source code in a way that could affect the market in the way the Sun/Oracle claims it has been affected. As such, Android's alleged impact on the market for Java was made possible by Sun/Oracle's decision to release OpenJDK; by making OpenJDK available for free via an open-source license, and not requiring parties to use all of the API packages, Sun/Oracle made it possible for anyone who was willing to put in the time and effort to transform the 37 API packages into something suited for the mobile market to impact the market in the same way it now complains Android impacted the market.

266. Based on public statements by Sun around the time of OpenJDK's release, the impact on the market for Java by releasing OpenJDK as a free, open-source implementation of Java SE was very much intentional on Sun's part. As noted by Sun's Chief Open Source Officer Simon Phipps, in a blog post on August 16, 2008, Sun understood that by making Java free and open source, the Java community would actually "benefit from the extended deployment range, the greater pool of expertise, and the greater diversity of interests that [would] result."[17] In a comment on Mr. Phipps' blog post, Sun's then-CEO Jonathan Schwartz agreed, expressly noting that the market for Java should not be limited to one proprietary licensing model: "Thank you for articulating with such clarity exactly what we're after – there's no one license for the planet, no one OS, nor just one rationale or audience. What matters most to everyone is…choice."[18] Mr. Schwartz also publicly stated at the 2006 Java One conference that Sun understood and intended that actual or potential competitors of Sun could use OpenJDK to create products that competed with Sun's own offerings. GOOG-00000383 ("We're doing this [OpenJDK] for a couple of reasons. One, it will drive community, and community is at the center of everything we're doing. It's not okay to ship an esoteric piece of technology and hope to make money off of it. That's not our strategy. Our strategy is to drive the network effect, to drive community, to drive volume in the marketplace so that we, along with our partners, and frankly, some of our competitors can go drive value in the marketplace.").

267. Besides the form of the license, Google did nothing by way of its use of the 37 Java API packages from the Apache Harmony project that it could not have done if it had instead used those same 37 Java API packages from OpenJDK.

[17] GOOG-00597926 (at -927).
[18] GOOG-00597926 (at -927).

268. As noted above, OpenJDK is licensed under the GPL-2.0-CE license. The Classpath exception provides as follows:

> Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License cover the whole combination.
>
> As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on this library. If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version.

http://www.gnu.org/software/classpath/license.html.

269. Programmers write programs in the Java programming language for Android or the Java platform. These programs typically make many calls of methods in the 37 API packages, *e.g.*, using java.lang.Math.sqrt for example. A compiler translates the Java program written by a user into bytecode, *e.g.*, for a Java Virtual Machine, a Dalvik virtual machine or the Android Runtime (ART). In translating the Java program, the compiler must determine if the API calls are valid, *e.g.*, is the method name spelled correctly, are the arguments passed properly, is the return value used appropriately. The compiler can determine whether the usage of the API is correct simply by knowing the API method signature. When the program executes, the implementation of the relevant API, *e.g.*, the code written to implement the functionality of Math.sqrt, must be available to the virtual machine or runtime environment. The process of including the implementation of the APIs with the program that calls the method in the API is termed linking. A program that calls Math.sqrt cannot execute unless the code implementing the

functionality of Math.sqrt is available to the runtime environment or virtual machine. The process of creating an executable program includes the process of linking a library, or implementation, to the code that calls a method in the library. Thus, linking to a library is something commonly done in Java, and—per the Classpath exception—does not mean that programs using any particular API must themselves be licensed under the GPL license.

270. There is also evidence that Android positively affected the market for Java. Java is the most popular programming language in the world. Smith Dep. Tr. at 32:9-15, 33:3-14; Ringhofer Dep. Tr. at 233:18-23; *see also* GOOG-10002044 (at -48); GOOG-00000379; GOOG-00000454; GOOG-00000513; GOOG-10002075 (at -76); GOOG-00000420. There are over 10 million Java developers in the world today. *Id.* The number of Java developers has increased over time. Smith Dep. Tr. at 32:17-33:1. The increase in and continued popularity of the Java programming language is due in large part to the fact that developers can use the Java programming language to write programs for Android. *See* GOOG-00000379; GOOG-00000454; GOOG-00000513; GOOG-10002075 (at -76); GOOG-00000420.

E. Other Considerations Related to Fair Use

271. As a practical matter, developers typically write code and software interfaces so that they are efficient, and comport with industry demands and widely accepted practices. They also often choose organizational strategies used in systems that are in the public domain. The selection, arrangement and organization of the APIs at issue reflect these practices.

272. Re-implementation of APIs is a common practice across languages and platforms. The C++ language and libraries have been used and updated since the first C++ reference guide was published in 1985. Many groups or projects contributed implementations of the libraries specified in the standard and continue to do so as the standard has been revised and updated. These include the GNU project, LLVM, Boost, Microsoft and others. Each projects have

different licenses, but for the core C++ libraries they each provide a different implementation of the same, standard APIs.

273. OpenGL is a wide-spread API for rendering 2D and 3D graphics. The functions specified by the API are typically re-implemented for new hardware platforms and for many different programming languages. Just as Java programs can call native C and C++ programs, language "bindings" are written in Java, C++, Javascript, and nearly every widespread language to be able to use the OpenGL APIs "natively," that is running on different hardware devices.

274. A typical web experience requires that browsers run JavaScript, or more formally ECMAScript, and the core libraries the language requires to run. Several re-implementations of the Javascript APIs are used by different browsers, e.g., Firefox, Chrome, and Microsoft's Edge/Internet Explorer. Mozilla, the foundation that oversees Firefox, has implemented JavaScript engines in both C++ and Java

1. Other implementations of Java SE.

275. As discussed above, besides the Sun/Oracle implementation of Java SE, there were various other implementations of Java SE in existence at or around the time Google developed and released Android. Specifically,

- **OpenJDK**: open-source implementation supported by Sun/Oracle;
- **GNU Classpath**: developed by the non-profit GNU Project; and
- **Apache Harmony**: developed by the non-profit Apache Foundation.

276. Sun/Oracle was aware of these other implementations of Java SE, including GNU Classpath and Apache Harmony.[19] In fact, Oracle incorporated GNU Classpath into a number of its commercial products:

[19] RT 1972:8-1974:12 (GNU Classpath); RT 2034:16-2036:24 (Apache); *see also* TX 2341.

277. Sun never publicly suggested that GNU Classpath had done anything wrong by developing and publicly distributing GNU Classpath, nor did Sun pursue legal action against GNU Classpath.[20] To the contrary, Sun/Oracle openly applauded and encouraged GNU Classpath and others to distribute its implementation of Java SE.[21] In fact, Oracle encouraged and promoted GNU Classpath by using code derived from GNU Classpath in Oracle's own products, at least some of which were commercial, and Sun promoted the use of code derived from GNU Classpath (such as the GCJ compiler) at its annual JavaOne conference as early as 1999. Sun also collaborated with members of the GNU Classpath community and the related IcedTea project in order to create Sun's own open source re-implementation of Java SE in OpenJDK.[22]

278. Similarly, Sun/Oracle was aware of the Apache Harmony project at least as early as 2005, but never suggested that Apache had infringed Sun's copyrights.[23] Sun's public position about Apache Harmony was that, as long as Apache did not use the Java trademark in connection with its implementation of the Java APIs, Apache was free to distribute Harmony for any purpose, including for use in mobile devices.[24] Oracle itself was a strong supporter of the Apache Harmony for several years before it acquired Sun.

279. Other supporters of Apache Harmony, such as IBM, have openly developed commercial software using Apache Harmony implementations of the Java APIs. IBM in particular has incorporated Apache Harmony implementations of several Java API packages into both versions 6 and 7 of its Java SE SDK, which is required to run IBM software written in

[20] RT 1973:24-1974:8; RT 1863:20-1864:2 (Oracle Response to Google RFA No. 145).
[21] GOOG-00000383.
[22] GOOG-00000221 (at -224-25); GOOG-00000316 (at -319-20); GOOG-00598060 (at -062); *see also* TX 2259 at 7-8.
[23] RT 561:25-562:1; RT 562:17-563:2; RT 1863:20-1864:2 (Oracle Response to Google RFA No. 145).
[24] TX 2341 (May 9, 2007 article quoting Sun CEO Jonathan Schwartz saying "there is no reason that Apache cannot ship Harmony today. . . ."); RT 2010:5-12.

Java.[25] These Apache Harmony implementations contain the same selection, arrangement and organization of many of the Java APIs at issue in this case.[26] IBM has shipped the versions of the IBM Java SE SDK containing these Apache Harmony implementations of the Java APIs at issue with hundreds of commercial products offered by IBM, including WebSphere Application Server, Lotus Notes, IBM Commerce, and other Java-based products.[27] Sun/Oracle was aware of IBM's commercial use of Apache Harmony in its products and provided IBM a platform to promote its use of Apache Harmony at its JavaOne conference as recently as 2010.[28]

280. In my opinion, the fact of the existence of these other implementations and Sun's acceptance of their existence suggested to computer programmers generally that re-implementing Java APIs did not qualify as copyright infringement and/or was protected by the fair use doctrine (or, as many programmers believed at the time—and perhaps still today—that APIs are not copyrightable).

2. Sun's and Oracle's implementation of APIs from others.

281. Sun and Oracle have both engaged in the type of API re-implementation that they now allege does not qualify as fair use. Specifically, Sun and Oracle have, in the past, used in their own products APIs originated by other companies.

a) Sun implemented and distributed APIs from previous generations of spreadsheets as part of Staroffice and Openoffice.org.

282. Between 1999 and 2011, Oracle, and Sun Microsystems before it, developed and distributed the StarOffice and OpenOffice.org "Calc" spreadsheet software, and this software implemented and distributed APIs from previous generations of spreadsheets created by other

[25] Duimovich Dep. Tr. at 38:12-39:3, 69:24-71:18, 74:17-21, 103:7-14, 121:14-24; GOOG-10002049; Ex. 1410; Ex. 1411; Ex. 1412.
[26] Duimovich Dep. Tr. at 42:11-15, 56:2-18, 71:22-72:3, 72:16-23, 116:23-120:1, 121:1-5; Ex. 1411; Ex. 1412.
[27] Duimovich Dep. Tr. at 39:4-40:13, 76:7-77:25, 81:24-82:3, 102:6-103:5; GOOG-10002049.
[28] Duimovich Dep. Tr. at 43:18-44:17, 45:10-19, 47:3-15, 48:5-11, 104:18-105:5; GOOG-10002049; Ex. 1410.

companies. The Calc spreadsheet software contains an API, and this API is in large part based on the APIs originally developed for older spreadsheet software, including Visicorp's Visicalc and Microsoft's Excel spreadsheet software. The implementation of the APIs in the Calc spreadsheet program allows spreadsheet models developed in Excel, for example, to also be useful and run in the StarOffice or OpenOffice programs.

283. Every spreadsheet program provides "spreadsheet functions" that enable users to write small programs—called "macros"—that manipulate data in a spreadsheet cell. For example, the function "ABS" calculates the absolute value of the number in a given spreadsheet cell, the function "AVERAGE" calculates the average value of the numbers in multiple spreadsheet cells, and "NPV" returns the net present value of an investment. These functions or macros are used by people using spreadsheets to create models whether these people are professional software developers, engineers, lawyers, investment bankers, scientists, or hobbyists. These functions and the macros they are used by constitute an API, because they are a mechanism that allows creation of written programs that communicate with the spreadsheet software's functionality.

284. Spreadsheets created by different software companies frequently use function names and argument structures from older spreadsheet programs. For example, the first column of the following table shows the names of all the spreadsheet functions that were supported in VisiCalc in 1979—VisiCalc's API. The other columns of the table show the function names used to operate the same functionality in Lotus 1-2-3, Microsoft Excel, and OpenOffice Calc. (The additional API elements or functionalities added in the later programs are not shown in this table.) As the chart shows, the function names originally used by VisiCorp's VisiCalc in 1979 were then used by Microsoft Excel 2003 and Oracle's OpenOffice.org. (Oracle subsequently

contributed OpenOffice.org to the Apache Software Foundation.) This shows that the original VisiCalc API of 1979 was included in the Microsoft and Oracle products, with only one exception (VisiCalc's "ERROR" function, which has been replaced by #N/A or #VALUE in Excel and Calc).

VisiCalc	Microsoft Excel	OpenOffice.org Calc
@ABS	ABS	ABS
@ACOS	ACOS	ACOS
@ASIN	ASIN	ASIN
@ATAN	ATAN	ATAN
@AVERAGE	AVERAGE	AVERAGE
@COS	COS	COS
@COUNT	COUNT	COUNT
@ERROR	#N/A or #VALUE!	#N/A or #VALUE!
@EXP	EXP	EXP
@INT	INT	INT
@LN	LN	LN
@LOG10	LOG10	LOG10
@LOOKUP	LOOKUP	LOOKUP
@MAX	MAX	MAX

VisiCalc	Microsoft Excel	OpenOffice.org Calc
@MIN	MIN	MIN
@NA	NA	NA
@NPV	NPV	NPV
@PI	PI	PI
@SIN	SIN	SIN
@SQRT	SQRT	SQRT
@SUM	SUM	SUM
@TAN	TAN	TAN

285. Attached as Exhibit C is a table showing the names of the spreadsheet functions in Microsoft Excel 2003 and OpenOffice.org Calc, which I prepared in 2011 based on the publicly available documentation for Microsoft Office Excel 2003, formerly available at http://office.microsoft.com/en-us/excel-help/excel-functions-by-category-HP005204211.aspx, and for OpenOffice.org Calc at http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category. Five rows from Exhibit C are reproduced here for discussion purposes:

Microsoft Excel	OpenOffice.org Calc
AMORLINC	AMORLINC
AND	AND

	ARABIC
AREAS	AREAS
ASC	
ASIN	ASIN

286. Exhibit C shows that many of the functions that constitute the API of Microsoft Excel 2003 were also implemented in OpenOffice.org Calc. Functions on the left are implemented in Excel, and functions on the right are implemented in Calc. In this sample from Exhibit A, three of the functions (AMORLINC, AREAS, and ASIN) are implemented in both spreadsheets, while ASC is implemented only in Excel and ARABIC is implemented only in StarOffice.

287. As shown in Exhibit C, overall, of the 340 functions implemented in the Excel 2003 spreadsheet function API, 324 (95%) were also implemented in StarOffice.

b) Sun implemented and distributed APIs from Linux as part of Solaris

288. Since 1999, the Solaris operating system, developed and distributed by Oracle, and Sun Microsystems before it, has contained or been delivered with APIs that are based on the APIs originally developed by the developers of the Linux operating system. The implementations of these APIs in Solaris facilitates the use of programs developed in Linux environments to run on Solaris machines.

289. The BrandZ project, also known as Solaris Containers, was a software system that Sun implemented starting in 2004. BrandZ worked with other software, called a "brand," to translate a non-Solaris operating system's functionality into the Solaris functionality, so that software written for the other operating system would run on Solaris. Essentially, each brand

helped "translate" communications that used the other operating system's APIs into communication with similar Solaris APIs. In particular, Sun developed a brand called the "LX Brand." The purpose of the LX Brand software was to "enable[] Linux binary applications to run unmodified on Solaris" ("BrandZ WebHome," *formerly available at* http://hub.opensolaris.org/bin/view/Community+Group+brandz/WebHome). To achieve this goal, several components of the Linux API are implemented by the LX Brand software, including signals, system calls, and the "/proc" interface. (*See* "BrandZ Overview," *formerly available at* http://hub.opensolaris.org/bin/download/Community+Group+brandz/WebHome/brandzoverview.pdf)

290. For example, the "/proc" interface allows programs to interface with the Linux operating system by reading and writing the contents of files in a special directory called "/proc." Reading and writing these files allows a program to discover the status of the operating system and processes running on the operating system. A process can be a program or a part of a program that the user is running and it can be part of the operating system, *e.g.*, it might facilitate communication over the Internet, with a printer, or allow one program to pass data to another program. In Linux environments and many Unix environments every process has a number that identifies it, the so-called Process Identifier or PID. Processes also have names—for example the process that starts up the first process for the operating system has PID one, but the name 'init' and a process designated for cryptographic programs might have the name 'crypto' and would certainly have a different PID than the 'init' process. One representative element of the /proc interface, known as "/proc/[pid]/status," allows a program to communicate with the operating system about the status of a particular process. To initiate the communication, the program asks

for the contents of the /proc/[pid]/status file—that is a file whose name is 'status' that is located in the directory/folder corresponding to the process identifier of a process, *e.g.*, /proc/523/status is the file that gives the status of process 523. The operating system responds to a request for information about a particular process by filling the file named 'status' with text that shows the status of the process. After such a request, the first eight lines of the /proc/[pid]/status file might, for example, look like this:

```
Name:   [the name of the process]
State:  [the status of the process]
Tgid:   [the "Thread Group ID" of the process]
Pid:    [the "Thread ID" of the process]
PPid:   [the "Thread ID of the process's parent]
TracerPid:      [The "Thread ID" of the tracing process]
Uid:    [ID numbers of users involved in the process]
Gid:    [ID numbers of groups involved in the process]
```

291. The text on the lefthand side of the file (such as "Name: ") is part of Linux's API. These are always present in /proc/[pid]/status. The text on the right is the information about the specific process, and will be different each time /proc/[pid]/status is accessed. Changes to this layout (for example, changing "Name" to "ID" or "Reference") would break applications that use this API. As a result, if another operating system wanted to be compatible with this API, it would need to print "Name:," "State:," etc., in exactly the same manner as Linux prints it.

292. The following chart shows the values—taken directly from the respective publicly available source code—which Linux and Sun's LX Brand use to create the text on the left hand side of the /proc/[pid]/status file. In each entry in the chart, "\t" means "tab," "\n" means end of line, and the "%s" is replaced by the relevant information for the particular process, so that "Name:\t%s\n" becomes

```
Name:       [the name of the process]
```

when the /proc/[pid]/process file is accessed.

Linux	LX Brand
"Name:\t%s\n"	"Name:\t%s\n"
"State:\t%s\n"	"State:\t%s\n"
"Tgid:\t%d\n"	"Tgid:\t%d\n"
"Pid:\t%d\n"	"Pid:\t%d\n"
"PPid:\t%d\n"	"PPid:\t%d\n"
"TracerPid:\t%d\n"	"TracerPid:\t%d\n"
"Uid:\t%d\t%d\t%d\t%d\n"	"Uid:\t%u\t%u\t%u\t%u\n"
"Gid:\t%d\t%d\t%d\t%d\n"	"Gid:\t%u\t%u\t%u\t%u\n"
"FDSize:\t%d\n"	"FDSize:\t%d\n"
"Groups:\t"	"Groups:\t"

293. Each line of the chart is identical, and this demonstrates that the output of the /proc/[pid]/status API is the same between Linux and the LX Brand software, and therefore that (in this respect, at least) Linux and the LX Brand software are compatible. If these lines were different, then the resulting file would be different, and the LX Brand software would not be compatible with Linux. Other elements of the /proc interface are similarly implemented in the LX Brand software.

294. The LX Brand software also re-implements Linux kernel system calls. System calls are a part of an operating system's API; they allow programs written by users to access resources managed by the operating system, e.g., to read and write files to kill processes, or to allocate memory to use in a program. These resources are managed by the operating system, but programs written by users to run on the operating system need access to some of the resources to be able to run properly or at all. The LX Brand software provides an emulation function which translates the Linux system call to an equivalent Solaris operating system call, for each of 317

Linux system calls. A relevant fragment of the publicly available source code that performs this translation, listing each Linux system call, and the LX Brand function that implements the system call, is attached as Exhibit D. This source code file indicates that there were 178 Linux system calls that were implemented as part of LX Brand (the other 139 system calls were either not supported or were able to directly use the equivalent Solaris system calls without translation). Each of the LX Brand implementations of the Linux system calls use the same name as the relevant Linux system call, with "lx_" prepended to distinguish them.

295. For example, the Linux system call "futex" was introduced to Linux beginning in 2002, and Solaris does not have a "futex" system call. In order to provide compatibility for Linux software running on the LX Brand, the LX Brand software provides an implementation of futex called lx_futex, which has essentially the same name as futex, takes similar arguments, and behaves similarly. The actual program logic that implements the LX Brand lx_futex function and the Linux system call are not similar, suggesting that they were independently created.

296. The 177 other system calls implemented by the LX Brand follow the same pattern: the Linux system call name, plus the lx_ prefix, is used to identify a function that takes similar arguments and behaves similarly to the Linux system call for which the function is named.

297. Sun's "LX Brand" software implements only a subset of the Linux operating system API, and so is not completely compatible with Linux. Sun's overview presentation states that it "support[s] a subset" of the /proc API and the "minimum needed" devices ("BrandZ Overview" at 23 and 24) and the design document notes that the "CLONE_PARENT" argument to the clone(2) system call is also only partially implemented (s*ee* "BrandZ Design Doc," section 3.5.1 ("Linux Threading"), *formerly available at*

http://hub.opensolaris.org/bin/view/Community+Group+brandz/design). This partial implementation still aids compatibility and programmer efficiency, because it is still better for the programmer to use some of the APIs than to have to completely rewrite the software to use new APIs.

298. Solaris has also incorporated specific APIs from the Linux C Library ("glibc") into the Solaris C Library. For example, the "uucopy()" system call, according to Sun's BrandZ Design Doc, "seems to be generically useful, so the uucopy() will be implemented in [Solaris] libc" and, in fact, Solaris gained an implementation of the uucopy system call in 2006, shortly after BrandZ was introduced (*see* Solaris's common/syscall/uucopy.c).

c) Oracle implemented and distributed APIs from IBM as part of the Oracle Database Server

299. As I will explain in this section, the Oracle Database server distributed by Oracle since 1979 contains an implementation of the API originally developed by IBM for the "System R" database.

300. The System R database's SQL API was first described in an academic paper published by IBM employees in 1974 ("SEQUEL: A Structured English Query Language," DD Chamberlin, et al.), and elaborated in a subsequent paper published in 1976.

301. The 1974 SEQUEL paper defined the following API elements or functionalities:

```
SELECT FROM
WHERE
GROUP BY
SUM
COUNT
AVG
MAX
MIN
```

302. IBM supplemented the functions in the 1974 paper in a subsequent paper published in 1976 ("System R: relational approach to database management," M. M. Astrahan et

al.), adding several new elements or functionalities to the SQL API: HAVING, ORDER BY, CURSOR, INSERT INTO, and DELETE.

303. Each of the API elements or functionalities referenced above, and defined in the 1974 and 1976 papers, were implemented by Oracle in 1979 and are still present in current releases of the Oracle Database server. Because these API elements are implemented in the Oracle Database server, a command using the API elements "SELECT FROM … WHERE …" would also be able to operate, with minimal changes, with current Oracle Database servers, as it did with the original IBM System R software (*see* "Oracle SQL: The Essential Reference," David C. Kreines (2000), Chapter 1, "Elements of SQL").

304. For example, because the Oracle system implements the API elements or functionalities defined in the 1974 paper, it will still execute commands written using the 1974 SEQUEL API. The 1974 paper gives this short command that uses elements defined in the 1974 paper:

```
SELECT NAME
FROM EMP
WHERE SAL
          SELECT    SAL
               FROM           EMP
               WHERE          NAME = Bl.MGR;
```

305. Because this command was written using API elements (in bold) originally defined by IBM but later implemented in the Oracle Database server, this command should still function in a modern Oracle Database server, and indeed, some sources report that this exact

command was used in early demonstrations of the Oracle database (*see* "Oracle SQL: The Essential Reference," David C. Kreines (2000), p. xiv and Chapter 1, "Elements of SQL").

3. Additional Oracle projects that support or encourage Android

306. From the moment it announced plans to acquire Sun, Oracle publicly encouraged Android. In June 2009, Oracle's then-CEO Larry Ellison publicly praised Google at the largest Java developer conference in the word, stating that Oracle was "flattered" by Android and expected to "see lots and lots of Java devices" "from our friends at Google." TX 2939.

307. Oracle has also encouraged its own customers or potential customers to use Android by creating and promoting products to run on Android phones. For example, Oracle promoted efforts to port its Java FX product to Android phones. *See* RT 1996:17-1997:25; TX 3103. Java FX is a user interface framework product licensed by Oracle. Smith Dep. Tr. at 64:18-23; *see also* DX 1321. Oracle also designed its Application Developer ("ADF") and Mobile Application Framework ("MAF") to enable its Java products to run on Android phones. Smith Dep. Tr. at 247:23-249:22; 251:2-254:10; *see also* DX 1321. Additionally, Oracle has created applications to run on Android phones, and it makes those applications available on Google Play. *See* https://play.google.com/store/search?q=Oracle&c=apps&hl=en (last visited January 8, 2016).

VII. CONCLUSION

308. I reserve the right to update and refine my opinions and analyses in light of any additional materials or information that may come to my attention in the future, including additional contentions by Oracle as well as any rulings issued by the Court in this case. I also reserve the right to supplement my opinions and analyses as set forth in this report in light of any expert reports submitted by Oracle and in light of any deposition or trial testimony of Oracle's experts.

Executed on the 8th of January, 2016 in Durham, NC.

Dr. Owen Astrachan

EXHIBIT A: OWEN ASTRACHAN CV

Owen L. Astrachan

Department of Computer Science
Box 90129
Duke University
Durham, NC 27708-0129
telephone: (919) 660-6522
email: ola@cs.duke.edu

January 8, 2016

I. Education

Ph.D.	Computer Science	Duke University	1992
M.S.	Computer Science	Duke University	1989
M.A.T.	Mathematics	Duke University	1979
A.B.	Mathematics	Dartmouth College	1978
	with distinction in Mathematics, Summa Cum Laude, Phi Beta Kappa		

II. Professional Appointments

Duke University, Department of Computer Science

Professor of the Practice
July 2000 —.

Associate Professor of the Practice
July 1996 — July 2000.

Director of Undergraduate Studies
September 1993 — .

Assistant Professor of the Practice of Computer Science
1993 — 1996.

Lecturer in Computer Science
1991–1992. Developed and taught the introductory course for non-majors. Served on lab committee determining priorities for physical improvements.

Research Assistant
1991 (June–Aug). Research Assistant at SRI International, AI group, working for Mark Stickel on the design of intelligent and efficient automated reasoning systems.

Research Assistant
1988–1991. Research Assistant for Donald W. Loveland, investigating automated theorem proving. Designed and implemented an OR parallel theorem prover that runs on a BBN Butterfly GP-1000, TC2000 and on a network of Sun workstations.

Senior Graduate Instructor
1986–1988. Solely responsible for developing the curriculum and teaching the first course for majors in the Computer Science Department. Assisted with course in Operating Systems.

Teaching Assistant
1985-1986 Served as Teaching Assistant for the the first two courses for majors in Computer Science. Responsible for designing laboratory exercises and running recitation sections.

University of British Columbia, Computer Science Department

Visiting Scholar and Lecturer
Sept 1998 — June 1999 (on sabbatical from Duke)

Experience in Secondary Education

Math and Computer Science Teacher Durham Academy
1980–1985. Taught Advanced Placement Calculus, Advanced Placement Computer Science, Multivariable Calculus and Linear Algebra, Geometry, Introduction to Computer Programming, Finite Mathematics. Developed curriculum for Finite Math, AP Computer Science, and Multivariable Calculus.

Math Teacher Camp Lejeune HS, Camp Lejeune, NC
1978–1980. Taught honors Trigonometry, honors Geometry, Algebra I, Pre-Algebra.

III. Honors

2007, NSF, CISE Distinguished Education Fellow, *Interdisciplinary Problem- and Case-based Computer Science*, one of two inaugural CDEF awardees (see grants).

2004, IBM Faculty Award, *Issues in Large-scale Software Componentization*

2003, ACM International Collegiate Programming Contest (ICPC) Coaches Award.

2002, Richard K. Lublin Award for Distinguished and Motivating Teaching

2002, Nominated for Alumni Distinguished Teaching Award

2001, Nominated for Alumni Distinguished Teaching Award

1998, Outstanding instructor of Computer Science, University of British Columbia (teaching on sabbatical)

1997, NSF Career Award

1996, Nominated for Alumni Distinguished Teaching Award

1995, Duke University, Trinity College of Arts and Science: Robert B. Cox Distinguished Teaching in Science Award

1995, Sigma Xi

1994, Nominated for Alumni Distinguished Teaching Award

1978, A.B. degree awarded with distinction, summa cum laude, Phi Beta Kappa

IV. Publications

Journals:

Steve Wolfman, Owen Astrachan, Mike Clancy, Kurt Eiselt, Jeffrey Forbes, Diana Franklin, David Kay, Mike Scott, and Kevin Wayne. Teaching-Oriented Faculty at Research Universities. *Communications of the ACM*. November 2011, v. 54. n 11. pp. 35-37.

Owen Astrachan and Robert Dewar. CS Education in the U.S.: Heading in the Wrong Direction. *Communications of the ACM*. July 2009, v. 52, n. 7, pp. 41-45.

O.L. Astrachan and D.W. Loveland. The Use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning*, v. 19 n.1, August, 1997, pp. 117-141.

Owen Astrachan, Kim Bruce, Robert Cupper, Peter Denning, Scot Drysdale, Tom Horton, Charles Kelemen, Cathy McGeoch, Yale Patt, Viera Proulx, Roy Rada, Richard Rasala, Eric Roberts, Steven Rudich, Lynn Stein, Allen Tucker, Charles van Loan. Strategic Directions in Computer Science Education. *ACM Computing Surveys*. v 28, n 4, December 1996.

O.L. Astrachan. METEOR: Exploring Model Elimination Theorem Proving. *Journal of Automated Reasoning*. v.13 n.2, 1994, pp. 283-296.

Books:

Owen Astrachan. *A Computer Science Tapestry: Exploring Programming and Computer Science with C++, Second edition*. McGraw-Hill, 2000.

Owen Astrachan. *A Computer Science Tapestry: Exploring Programming and Computer Science with C++*. McGraw-Hill, 1997.

Owen Astrachan. *The Large Integer Case Study in C++*. The College Board, Advanced Placement Program, 1997.

Book Chapters:

Owen Astrachan and Robert Duvall and Eugene Wallingford. Bringing Extreme Programming to the Classroom. in *Extreme Programming Perspectives*, Giancarlo Succi (ed.), Addison Wesley, 2002.

O.L. Astrachan and D.W. Loveland. METEORs: High Performance Theorem Provers Using Model Elimination. in *Automated Reasoning: Essays in Honor of Woody Bledsoe* ed. R.S. Boyer, Kluwer Academic Press 1991.

Miscellaneous:

O.L. Astrachan and Susan Horwitz and the Advanced Placement Computer Science Development Committee. *Communications of the ACM*. Technical Opinion: The First Course Conundrum. June, 1995. Pages 117-118.

Refereed Conferences:

Owen Astrachan, R. Brook Osborne *et al.* Computer Science Principles: Analysis of a Proposed Advanced Placement Course. *SIGCSE Technical Symposium on Computer Science Education*, Denver, CO, 2013

Owen Astrachan. Pander to Ponder, *SIGCSE Technical Symposium on Computer Science Education*, Chattanooga, TN, 2009

Casey Alt, Owen Astrachan, Jeffrey Forbes, Richard Lucic, and Susan Rodger. Social Networks Generate Interest in Computer Science. *SIGCSE Technical Symposium on Computer Science Education*, Houston, TX, 2006.

Owen Astrachan. Non-Competitive Programming Contest Problems as the Basis for Just-in-time Teaching. *Proceedings of Frontiers in Education*, October 2004.

Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. *SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, 2003.

Owen Astrachan and David Bernstein and Andrew English and Benjamin Koh. Development Issues for a Networked, Object Oriented Gaming Architecture (NOOGA) Teaching Tool. *Proceedings of Frontiers in Education*, November 2002.

Owen Astrachan and Robert Duvall and Jeffrey Forbes and Susan Rodger. Active Learning in Small to Large Courses *Proceedings of Frontiers in Education*, November 2002.

Owen Astrachan and Robert Duvall and Eugene Wallingford. Bringing Extreme Programming to the Classroom, *Proceedings of XPUniverse*, Raleigh, NC, July, 2001.

Owen Astrachan. OO Overkill: When Simple is Better than Not, *SIGCSE Technical Symposium on Computer Science Education*. Charlotte, NC, February 2001.

Charles Keleman, Allen Tucker, Peter Henderson, Kim Bruce, Owen Astrachan. Has Our Curriculum Become Math-Phobic?, *SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)*, June 2000.

Owen Astrachan and Eugene Wallingford. Loop Patterns. *Pattern Languages of Programming (PLoP)*, Allerton Park, IL, August, 1998.

Owen Astrachan. Hooks and Props as Instructional Technology. *SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)*, August 1998.

Owen Astrachan, Geoffrey Berry, Landon Cox and Garrett Mitchener. Design Patterns: An Essential Component of CS Curricula. *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, February 1998.

Owen Astrachan and Susan Rodger. Animation, Visualization, and Interaction in CS 1 Assignments, *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, February 1998.

Owen Astrachan and Robert Smith and James Wilkes. Application-based Modules using Apprentice Learning for CS 2. *SIGCSE Technical Symposium on Computer Science Education*. San Jose, CA, February 1997, pp 233–237.

Owen Astrachan and Trevor Selby and Joshua Unger. An Object-Oriented, Apprenticeship Approach to Data Structures using Simulation. *Frontiers in Education*, Salt Lake City, Utah, 1996, pp 130–134.

Owen Astrachan and David Reed. AAA and CS1 : The Applied Apprenticeship Approach to CS 1. *SIGCSE Technical Symposium on Computer Science Education*. Nashville, TN, March 1995.

Owen Astrachan and Claire Bono. Using simulation in an objects-early approach to CS1 and CS2. *OOPSLA Conference Proceedings, Educator's Forum*. Portland, Oregon, October 1994.

O. L. Astrachan and D.W. Loveland. *METEOR*: Model Elimination Theorem Proving with Lemmas (system abstract). *Twelfth Conference on Automated Deduction* (CADE-12). Nancy, France, 1994.

Owen Astrachan. Self reference is a Thematic Essential. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, March 1994.

Owen Astrachan. METEOR: Exploring Model Elimination Theorem Proving. *Workshop on Theorem Proving with Analytic Tableaux and Related Methods*. Marseilles, France, April 1993.

Owen L. Astrachan, Vivek Khera, and David Kotz. The Duke Internet Programming Contest: A Report and Philosophy. *SIGCSE Technical Symposium on Computer Science Education*. Indianapolis, IN, February 1993.

Owen L. Astrachan and Mark E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. *Eleventh Conference on Automated Deduction* (CADE-11). Saratoga Springs, NY, June 1992.

Owen Astrachan. Finding a Stable roommate, job or spouse: a case study crossing the boundaries of Computer Science Courses. *SIGCSE Technical Symposium on Computer Science Education*. Kansas City, MO, March 1992.

Owen Astrachan. Pictures as Invariants. *SIGCSE Technical Symposium on Computer Science Education*. San Antonio, TX, March 1991.

Owen Astrachan. METEOR: Model Elimination Theorem Prover for Efficient OR-Parallelism. *AAAI Spring Symposium on Representation and Compilation in High Performance Theorem Proving: Titles and Abstracts*, ed. W.W. Bledsoe, M. Stickel, P. Lincoln, R. Overbeek, and D. Plaisted, Stanford, CA, March 1989.

Unrefereed Reports:

Owen L. Astrachan and Donald W. Loveland *The Use of Lemmas in the Model Elimination Procedure*. Duke University Technical Report CS-1993-25.

Owen L. Astrachan. *METEOR: Exploring Model Elimination Theorem Proving*. Duke University Technical Report CS-1992-22.

Owen L. Astrachan. *Investigations in Model Elimination Based Theorem Proving*. Ph.D. Thesis. Also Duke University Technical Report CS-1992-21.

Owen L. Astrachan and Mark E. Stickel. *Caching and Lemmaizing in Model Elimination Theorem Provers*. SRI International Technical Note 513, December 1991.

v. Service

Professional Service

2015 Program Committee, Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), IEEE, Charlotte, NC.

2015 Program Committee, ICER: International Computer Science Education Conference.

2014– K-5 Program Affiliate for Code.org Professional Development.

2014 Program Committee, ICER: International Computer Science Education Conference.

2013 Program Committee, ICER: International Computer Science Education Conference.

2013– Code.org Educational Advisory Council.

2009 – 2012*College Board* HEAC: Higher Education Advisory Committee for Advanced Placement. Provide oversight and advice regarding the program.

2009, July, NSF Review Panel: Broadening Participation in Computing (BPC)

2009 *NITRD: Networking and Information Technology Research and Development Program*, panelist at public forum for discussion of the 2009 Federal Strategic Plan

2008 – *NSF/College Board* joint group on the First Year in Computer Science (chair).

2008, May, NSF Review Panel *CPATH*

2007 – *College Board AP Computer Science Redesign Commission* Committee charged with examining and redefining the Advanced Placement Computer Science Program.

2006– ACM Ed-Council. One of 25 members providing leadership and governance to the ACM about educational activities and outreach.

2005, July, NSF Review Panel *Advanced Learning Technology*

Program Committee OOPSLA 2005, Educator's Symposium

Program Committee OOPSLA 2004, Educator's Symposium

Internet & Society Idea Exchange Faculty Steering Committee for courses related to Internet and Society (oversight from Harvard and MIT, including 55 faculty from around the world).

NSF Review Panel CRCD Program
2004, reviewed proposals for the CRCD program in Computer Science at NSF.

ACM/College Board Digital Library Project 2004 – Advisory Committee to develop a project supporting Compute Science in high schools as part of the national NSF-sponsored digital library program.

ACM/College Board JETT Steering Committee 2002 – , Member of four-person steering committee providing oversight for joint ACM/College Board committee reviewing and approving national sites to host high school outreach programs for computer science.

Illinois Math and Science Academy
2002 – 2003, Member of three-person external review board for Mathematics/Computer Science at IMSA.

College Board/High School Computer Science AP Computer Science
2001 – 2002, Member of College Board *ad hoc* professional development committee to develop standards for training/educating high school teachers and workshop consultants in computer science.

NSF Review Panel CISE Program
2000, reviewed proposals for the CRCD program in Computer Science at NSF.

Math and Computer Science Mathematical Association of America
1999. Committee made recommendations to the MAA on the role of mathematics in computer science. Committee consisted of Alan Tucker, Charles Keleman, Dale Skrien, Charles van Loan, Peter Henderson, Kim Bruce, Owen Astrachan.

Chair, Advisory Committee for AP Computer Science College Board

1999–2000. Committee making recommendations on the use of new languages and curricula in Advanced Placement Computer Science. Committe consists of David Gries, Robert (Corky) Cartwright, Henry Walker, Ursula Wolz, Cay Horstmann, Fran Trees, Rich Kick.

External Oversight Board North Carolina Central University

1997 – 1998, oversee the growth and accreditation of the Computer Science Department.

NSF Review Panel CISE Program

1997, reviewed proposals to the Education Innovation program in the CISE directorate of NSF.

External Review Committee Oberlin College

1996, Member of external review committee to evaluate computer science department at Oberlin.

Program Committee CADE-13

1995/6, Member of program committee for CADE-13, Conference on Automated Deduction.

Chair, Advisory Committee for AP Computer Science College Board

1995–1996. Committee makes recommendations on the best use of C++ on the Advanced Placement Exam. The Committee is convened by the College Board, with representation from SIGCSE, the Computer Science Education special interest group of the ACM.

Judge, ACM International Programming Contest Association for Computing Machinery

1994–1997. One of six people responsible for developing problems and judging solutions for the ACM Programming Contest finals.

Chief Reader, Advanced Placement Computer Science Educational Testing Service

1989–1994. Responsible for developing grading standards and assigning scores for the AP exam in Computer Science. Assist with the development of the exam. Oversee, hire, and manage 70 University faculty consultants and High School educators in the grading of 10,000 AP exams taken by secondary students throughout the world.

Member, AP Computer Science Development Committee The College Board

1985–1989. Responsible for developing curriculum and devising tests for the AP exam in Computer Science.

Director, Duke Internet Programming Contest

1990 — 1994 . Co-founded a computer programming contest held in real-time over the Internet, involving 60 teams from 37 institutions in 5 countries (1990); 240 teams from 100 institutions in 9 countries (1991); 290 teams from 140 institutions in 14 countries (1992), 495 teams from 200 institutions in 20 countries (1993). Developed the problems used in the contest, designed solutions for the problems, and co-directed the administration of the contest.

IEEE Programming Contest

1994–1995. Coach of the Duke undergraduate IEEE Programming team. This competition is by invitation only to sixteen teams throughout the world. In 1994 Duke participated for the first time. In 1995 Duke won the contest.

ACM Programming Contest

1993–. Coach of Duke undergraduate ACM Programming Team. In 1994 the team won the mid-Atlantic regional contest and placed third in the world (first U.S. team) in the world finals. In 1995 the team won the mid-Atlantic regional contest and advanced to the world finals, finishing 22nd. In 1997 the team advanced to the world finals. In 1998 the team advanced to the world finals. (On sabbatical in 1999-2000.) In 2001, the team advanced to the world finals. In 2002 Duke had four teams in the top fifteen, and the top two in the Midatlantic regional contest; the top team advances to the world finals. In 2003 Duke advanced to the world finals. In 2004 Duke won the region (tied for first), advanced to the world finals and had three teams in the top 15 of the region; in the world finals Duke was one of four US teams that placed (above honorable mention) and was tied with Caltech and MIT for second among US teams. In 2005 Duke won the Region and advanced to the world finals receiving an honorable mention. In 2006 a Duke team advanced to the world finals, in 2007 Duke won the region and participated in the world finals, in 2008 Duke received an at-large bid to the world finals (to be held in 2009).

1989–1990. Member of Duke Programming team, 1989 and 1990 ACM International Programming Contests. Finished fourth in 1989 (Louisville, KY) and eighth in 1990 (Washington, DC) contest (world) finals.

Referee Activities

2014	SIGCSE	Technical Symposium on Computer Science Education
2013	SIGCSE	Technical Symposium on Computer Science Education
2012	SIGCSE	Technical Symposium on Computer Science Education
2011	SIGCSE	Technical Symposium on Computer Science Education
2010	SIGCSE	Technical Symposium on Computer Science Education
2009	SIGCSE	Technical Symposium on Computer Science Education
2008	SIGCSE	Technical Symposium on Computer Science Education
2007	SIGCSE	Technical Symposium on Computer Science Education
2006	SIGCSE	Technical Symposium on Computer Science Education
2005	ITiCSE	SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)
2005	SIGCSE	Technical Symposium on Computer Science Education
2004	ITiCSE	SIGCSE Conference on Integrating Technology into Computer Science Education (ITiCSE)
2004	SIGCSE	Technical Symposium on Computer Science Education
2003	SIGCSE	Technical Symposium on Computer Science Education
2001	SIGCSE	Technical Symposium on Computer Science Education
2000	SIGCSE	Technical Symposium on Computer Science Education
1996	SIGCSE	Technical Symposium on Computer Science Education
1995/6	CADE-13	Conference on Automated Deduction
1995	TABLEAUX '95	Workshop on Theorem Proving with Analytic Tableaux and Related Methods
	IJCAI	International Joint Symposium on Artifical Intelligence Automated Reasoning Track
	SIGCSE	Technical Symposium on Computer Science Education
1994	SIGCSE	Technical Symposium on Computer Science Education
1993	ILPS	International Logic Programming Symposium
	IJCAI	Automated Reasoning track
	ISMIS	International Symposium on Methodologies for Intelligent Systems
	SIGCSE	Technical Symposium on Computer Science Education
1992	CADE-11	Conference on Automated Deduction
	SIGCSE	Technical Symposium on Computer Science Education
1991	SIGCSE	Technical Symposium on Computer Science Education
1990	CADE-10	Tenth Conference on Automated Deduction

Duke Service

2012-2013. Committee on Facilities and Environment.

2011-2013. Member of Academic Council.

2009. Chair promotion committee for Jeffrey Forbes.

2008. Member of Office Education Committee (OEC) overseeing appointments of ROTC faculty.

2007-2009. Member of Academic Council.

2007-2010. University Committee on Admissions and Financial Aid (Academic Council).

2007–2009 Member of QEP (quality enhancement plan) University Committee to create a 75-page documentn outlining Duke's future as part of our 10-year SACS re-accreditation process.

2008 Search Committee, Dean to replace Robert Thompson, resulted in appointment of Lee Baker.

2007. Chair promotion committee for Susan Rodger.

2007-2009. University Commencement Committee.

2006-2007. Chair *ad hoc* Committee to Distingish Trinity College Degrees. Will report on the status of the BA and BS degrees at Duke.

2006–. Member of Faculty Research Committee that decides on Trinity College competitive grants and awards to faculty for research and conference activity.

2005. Member of Commitee on Departmental Support for Technology as part of the University committeess on Strategic Directions.

2005. Member of Arts and Science search commitee for the A&SIST Associate Dean.

2004-2005. Chair re-appointment committee for Prof. Richard Lucic

2004-. ISIS (Information Sciences and Information Studies) Faculty Steering Committee.

2004-. One of three faculty overseeing the development of teaching and learning, in conjunction with the CIT, as part of re-allocation of resources regarding the former center for teaching and learning.

2004-2007. Member Executive Committee of the Arts and Science Council (ECASC).

2004-2007. Member of ITAC, Committee on Information Technology at Duke.

Chair re-appointment committee for Prof. Richard Lucic 2002.

Member re-appointment committee for Prof. Jeffrey Forbes 2002.

Member Advisory Board for BlackBoard at Duke, 2002–

CITIE, IT skills committee, 2002.

Member Academic Integrity Council, 2001–

Member Executive Committee of the Arts and Science Council (ECASC) 2000–2003.

Interviewed candidates for A.B. Duke program, 2000, 2001

Member Board of Directors for Center for Instructional Technology (CIT), 1999–.

Arts and Science Council, 1999–2008.

Chair, Arts and Science Committee on Integrated Cluster Classrooms, 1999–2000.

Chair re-appointment/promotion committee for Prof. Robert Duvall, 2000.

Chair, University Planning Group on Instructional Technology, 1999–2000.

Search Committee, head of Career Development Center, 1998.

Member B.N. Duke Scholarship Committee, 1997

Member Core Team on evaluating use of Instructional and Information Technology, 1997.

ITAC Committee on Student Computing, 1996.

Chair search committee for Lecturer position in Computer Science, 1996.

Chair re-appointment/promotion committee for Prof. Susan Rodger, 1996.

Search Committee, Assistent Dean of Student Development, 1996.

Arts and Science Council, 1995–1996.

Member of Management Team (Center for Teaching and Learning) to develop an Exercise in Interactive Theatre for "Developing Teacher Knowledge", 1996.

Director of Undergraduate Studies, 1993–.

Provost's ad-hoc committee for Computer Technology and Education, 1995–1997.

Member of Steering Committee, Schulzbeger Interactive Learning Laboratory, Teaching and Learning Center, 1994–1996.

Faculty Advisor, DULUG: Duke University Linux User's Group, 1995–.

Departmental major advisor 1992– (supervise 20 first majors, 19 second majors per year.)

Premajor advisor at Duke University, 1986– 1998(ten first year students each year, total of 20 per year)

Chaired search committee for Assistant Professor of-the-practice, 1994.

Committee on the role of teaching for graduate students 1990–1992

VI. Consultancies

Coursera Specialization

2015, created five course MOOC, Java Programming, An Introduction to Software Specialization, `https://www.coursera.org/specializations/java-programming`

Summit6 v. Motorola Mobility

2015 (February-June). Retained by Motorola and Google (Kilpatrick Townsend and Stockton), Northern Distict of Texas, in a patent infringement and invalidity case relating to mms-messaging and sending media over the Internet, 7:14-cv-00014, 7:14-cv-00160. Prepared both invalidity and non-infringement reports, case settled before deposition. Worked primarily with attorney Shayne O'Reilly, also with Jon Harris, Clay Holloway.

CA v New Relic

2013–2014. Retained by CA, Inc (Bracewell and Giuliani), Eastern District of New York, in a patent infringement case related to application performance monitoring, Case 12-cv-05468-JS-WDW. Prepared expert report, deposed in March 2014. Worked with attorneys Barry Shelton, Calvin Cheng, others.

Expert Witness, Greenberg Traurig

2011 — Retained by Google in "Oracle v Google: Complaint for Patent and Copyright Infringement: Java/Android", US District Court, Northern District of California. Prepared expert report, deposition in matters related to Java/Android APIs and copyright.

Expert Witness, Alston and Bird

2010 — Retained by Nokia in "Certain Mobile Communications and Computer Devices and Components Thereof," ITC Inv. No. 337-TA-704 (complaint filed Jan. 15, 2010), before the US International Trade Commission. Worked on Markman report, expert report, deposition and testimony.

Expert Witness, Alston and Bird

2008/2009 — Retained by Plaintiffs in Move, Inc., Nat'l Assoc. of Realtors, and Nat'l Assoc. of Home Builders vs. Real Estate Alliance Ltd et al., 2:07-CV-02185-GHK-AJW (filed Apr. 3, 2007) in the Central District of California. Worked on claim construction and expert report preparation.

Consultant, College Board

2008 – Oversee and help plan colloquium for college faculty (attended by 70 faculty) to understand current and future directions for AP Computer Science.

Google

2006 (six months) – Worked as an external contractor to help develop material used internally at Google for educating Google software engineers about Java and C++ programming.

Expert Witness, Womble, Carlyle, Sandridge and Rice

Software expert regarding work related to a contract dispute.

Consultant, AP Computer Science Educational Testing Service

1994 –. Advise development committee on incorporating C++ and Java into the Advanced Placement Program. Wrote Pascal/C++ case studies for use on the exam. Provided workshops for high school and college consultants in making transition to object oriented programming. Critique free response questions that are part of the national exam (2002, 2004).

Case-Study author, AP Computer Science Educational Testing Service

1994 –. Write the case study for use in the 1997-2000 AP exams. Write the code for the case study used in the 2001-2004 exams. A case is a "literate program", a treatise on the design, development, and implementation of a programming solution to a problem.

VII. Panels/Conference Activities

Advanced Placement Computer Science Principles (APCSP): A report from Teachers with R. Brook Osborne, SIGCSE Conference Memphis, TN, 2016.

Advanced Placement Computer Science Principles (APCSP): A report from Teachers with R. Brook Osborne, CSTA Conference, San Diego, CA, 2016.

Advanced Placement Computer Science: Two Courses, All Students, with Lien Diaz, CSTA Conference, San Diego, CA, 2016.

High School CS Breadth in Depth: CSP x ECS x APCSA, with Gail Chapman and Don Yanek. CSTA Annual Conference, Dallas, TX, 2015.

Computer Science Principles Curricula, with Dan Garcia, Bennett Brown, Jeff Gray, Ralph Morelli, Nigamanth Sridhar, and Baker Franke. CSTA Annual Conference, Dallas, TX, 2015.

Partners for a Path in K-12 Computer Science, with Pat Yongpradit, Gail Chapman, Joanna Goode, Emmanuel Schanzer, Kiki Prottsman, and Irene Lee. CSTA Annual Conference, Dallas, TX, 2015.

Computer Science Principles Curricula: On-the-ground, adoptable, adaptable, approaches to teaching, with Daniel Garcia, Bennett Brown, Jeff Gray, Ralph Morelli, Marie des Jardins, Calvin Lin, Nigamanth Sridhar, Bradley Beth. SIGCSE, Kansas City, MO, 2015.

Scaling High School Computer Science: Exploring Computer Science and Computer Science Principles with Ralph Morelli, Jeff Gray, Gail Chapman. SIGCSE, Kansas City, MO, 2015.

CSP Pilot Instructors: Update from the Classroom, with Brook Osborne, Lynn Norris, Crystal Furman, and Seth Pizzo. CSTA Annual Conference, St. Charles, IL, 2014.

Search: Programming and Computational Thinking CSTA Annual Conference, St. Charles, IL, 2014.

AP Computer Science Principles: A New AP Course and Exam, with Rich Kick, Lien Diaz. National Council of Teachers of Mathematics (NCTM), New Orleans, LA, April 2014.

Diverse Learners, Diverse Courses, Diverse Projects: Learning from Challenges in New Directions, with Brook Osborne, Jeff Gray, Irene Lee, Bradley Beth, SIGCSE, Atlanta, GA, March 2014.

A Public/Private Partnership for Expanding Computer Science in Schools, with Amy Briggs, Brook Osborne, Pat Yongpradit, Gail Chapman, Joanna Goode, SIGCSE, Atlanta, GA, March 2014.

Education for a Cyber-Capable Citizenry: From Majors to Nonmajors, From Ethics to Understanding, with Brook Osborne, American Association of Colleges and Universities (AACU), Transforming STEM Education, October 2013, San Diego, CA

CS Principles — Piloting a Portfolio Assessment, CSTA Conference, Boston, MA, 2013, with Brook Osborne, Rich Kick, Rebecca Dovi, Baker Franke, Deepa Muralidhar

A New AP Course: Computer Science Principles, with Lien Diaz, Amy Briggs, Jill Kushner, Andrew Kuemmel, Deepa Muralidhar, AP Annual Conference (College Board), Las Vegas, NV, July 2013.

Learning with Exploring Computer Science: Connections to AP Computer Science with Lien Diaz, Amy Briggs, Gail Chapman, Joanna Goode, Jody Paul, AP Annual Conference (College Board), Las Vegas, NV, July 2013.

CS Principles: Development and Evolution of a course and Community, SIGCSE, Denver, CO, 2013, with Amy Briggs, Lien Diaz, Brook Osborne.

Broadening Participation: ECS and CS Principles, CS&IT, Irvine, CA, 2012, with Gail Chapman and Brook Osborne.

CS Principles: A new Course in Computer Science, North Central Business Education Assocation, 2012, Indianapolis, Indiana, with Brook Osborne.

Update on the CS Principles Project, SIGCSE, Raleigh, NC, 2012, with Amy Briggs, Jan Cuny, Lien Diaz, Chris Stephenson.

CS Principles: Piloting a National Course II, SIGCSE, Raleigh, NC, 2012, with Amy Briggs, Ralph Morelli, Dwight Barnet, Jeff Gray.

CS Principles: Piloting a New Course at National Scale, SIGCSE, Dallas, 2011 (with Larry Snyder, Tiffany Barnes, Dan Garcia, Jody Paul, Beth Simon).

The CS10K Project: Mobilizing the Community to Transform High School Computing SIGCSE, Dallas, 2011 (with Jan Cuny, Chris Stephenson, and Cameron Wilson)

The CS/10K Project, CRA/Snowbird Conference, Snowbird UTAH, July, 2010.

Code as a Metaphor for Computational Thinking, CSTA/CSIT Symposium, Google, Mountain View, CA, July, 2010.

Re-imagining the First Year of Computer Science, SIGCSE, Milwaukee, WI, 2010 (with Lien Diaz, Chris Stephenson, Jan Cuny, Amy Briggs)

FOSS Workshop, Free and Open Source Software, SIGCSE, Chattanooga, TN, 2009, invited speaker.

Computational Thinking Panel, SIGCSE, Chattanooga, TN, 2009 (with Amber Settle, Susanne Hambrusch, and Joan Peckham.

Advanced Placement Computer Science: The Future of Tracking the First Year of Instruction, Special Session, SIGCSE, Chattanooga, TN, 2009 (with Henry Walker, Chris Stephenson, Lien Diaz, and Jan Cuny)

Nifty Assignments, Special Session, SIGCSE, Chattanooga, TN, 2009 (with Nick Parlante)

Innovating our Self Image Special Session, SIGCSE, Portland, OR, 2008 (with Peter Denning)

Teaching Tips We Wish They Told Us Before We Started Special Session, SIGCSE, Cincinnati, OH, 2007 (with Dan Garcia, Nick Parlante, Stuart Reges).

Resolved: Objects Early Has Failed SIGCSE, St. Louis, 2005, Special Session (with Stuart Reges, Kim Bruce, Michael Kölling, Elliot Koffman).

But it Looks Rights: Bugs Students Don't See SIGCSE, Norfolk, 2004, Special Session (with David Ginat, Daniel Garcia, Mark Guzdial).

Colorful Illustrations of Algorithmic Design Techniques SIGCSE, Charlotte, 2001, Special Session (with David Ginat, Joseph Bergin, Dan Garcia).

Nifty Assignments in CS1 and CS2, Panelist, SIGCSE, Charlotte, 2001 (with Michael Clancy, Nick Parlante, Rich Pattis, Stuart Reges, Julie Zelenski).

FYI 2000: First Year Instruction, developed, organized, and chaired a workshop on first year instruction in computer science. The workshop had invited talks from both industry (Jon Bentley) and academia (Shriram Krishnamurthy, Richard Pattis) and was attended by more than 40 faculty from across the country. The workshop was sponsored by NSF, Microsoft, and the Duke Computer Science Department.

Patterns in Computer Science, (co-organizer with Eugene Wallingford), SIGCSE. Austin, TX. March 2000.

The Future of Advanced Placement Computer Science (panel). *SIGCSE Technical Symposium on Computer Science Education*. Austin, TX, 2000. With Corky Cartwright, David Gries, Cay Horstmann, Richard Kick, Fran Trees, Henry Walker, Ursula Wolz.

Nifty Assignments in CS1 and CS2, Panelist, SIGCSE, New Orleans, 1999 (with Michael Clancy, Nick Parlante, Rich Pattis, Stuart Reges, Julie Zelenski).

Incorporating Patterns into CS courses and *Writing Patterns for CS Courses* (co-organizer with Eugene Wallingford). SIGCSE, New Orleans, March 1999.

Future Directions in CS2 and Data Structures. Organized and ran the workshop that was held in conjunction with OOSPLA-98, Vancouver, CA, October 1998 (20 participants)

Object Oriented Design. Invited Participant, OOPSLA-97, Atlanta, Georgia, 1997.

Teaching Object-Oriented Programming: Practical Examples Using C++ and Java, Tutorial at PLDI 97, Las Vegas, Nevada, June 1997.

Future Directions in Data Structures and CS2. Organized and ran two-day workshop held at Duke co-sponsored by NSF, 32 participants, March, 2000.

Teaching Object-Oriented Design in the first year. Invited speaker and participant. OOPSLA-96, San Jose, CA, October, 1996.

Strategic Directions in Computing Research (SDCR), working group in Computer Science Education. Sponsored by ACM, CRA, and NSF, Boston, MA, June, 1996.

How to teach C++ in Introductory Courses, Tutorial part of PLDI, FCRC 1996, Philadelphia, PA, sponsored by SIGPLAN

Formal Methods Considered [Help — Harm]ful: Engaging students in the first year. Exploring Formal Methods in the Early Computer Science Curriculum, Joint NSF/US Department of Education Workshop. September 16, 1995 (invited speaker).

Developing an Object-Oriented Class Library. NSF sponsored workshop. Colgate University, June 1995. (invited participant)

A Sorcerer's Apprentice Approach to using C++ in CS1. NECUSE (New England Consortium on Undergraduate Science Education) Workshop in Introductory Computer Science Curricula. January 1995.

Measuring Performance of Automated Theorem Provers (with D. W. Loveland). *Twelfth Conference on Automated Deduction* (CADE-12), Workshop on Evaluation of Automated Theorem Proving Systems. Nancy, France, 1994.

Acquiring Object-Oriented Technology: A Bridge between Industry and Academia (invited participant). US West, Boulder Colorado, March 1994.

OOP: An introduction for Secondary School Teachers. Workshop delivered to secondary school computer science teachers in Dallas, TX, August 1993.

Simulation: A vehicle for exploring OOP. *Object-Oriented Curriculum Development Workshop*. NSF sponsored workshop. Colgate University, July 1993. (with Claire Bono)

A Tapestry of Fundamental Ideas and Concepts in Computer Science: a Programming, Contextural View for Liberal Arts Students *L**3: Logic, Loops, and Literacy*. NSF sponsored workshop on computer science for non-majors, Brooklyn College, May 1993.

Human Interaction with a High-Performance Theorem Prover. *International Joint Conference on Artificial Intelligence*. Workshop on Automated Theorem Proving. Chambery, France, 1993. (with D.W. Loveland)

Logic Programming Considered Harmful? *Joint Internatinal Conference on Logic Programming*. Prolog as a first language track, Washington DC, November 1993.

Caching to reduce redundancy in Model Elimination Theorem Provers. *Joint Japanese-American workshop in Automated Theorem Proving*, Argonne National Labs, June 1991.

The METEOR implementations of the Model Elimination procedure. *Workshop on Proof Theory and Automated Theorem Proving*. Oberwolfach, Germany, April 1991. (with D.W. Loveland)

Online Exams in Advanced Placement Computer Science. *National Council of Teachers of Mathematics Conference*. San Francisco, April 1984.

VIII. Invited Lectures, Talks, and Workshops.

CS Principles: Present and Future, with Brook Osborne, NSF CE21/Delaware Project, June 2013.

CS Principles: Present and Future, with Brook Osborne, NSF CE21/Alabama Project, June 2013.

CS Principles: Present and Future, with Brook Osborne, NSF CE21/Mobile Project, Hartford CT, July 2013.

CS Principles: Present and Future, with Brook Osborne, NSF REThink CS Project, Philadelphia, PA, July 2013.

Code: How to Use it, NSF REThink CS Project, Philadelphia, PA, July 2013.

Code as a Metaphor for Computational Thinking, Distinguished Faculty Lecture, Virginia Tech, February 24,2012

CS Principles, AP Annual Conference, July 2012, Orlando, FL.

CS Principles: Report and Progress, NCWIT National Summit, May, 2012.

CS Principles, AP Annual Conference, July 2011, San Francisco.

CS Principles: Report and Progress, NCWIT National Summit, May, 2011.

Developing a New, National Course in Computer Science, UNC Greensboro, January 25, 2011.

Code as a Metaphor for Computational Thinking, Harambeenet workshop, Durham, NC, July 2010.

CS Principles and the CS10K project, NSF, BPC Community Meeting, Los Angeles, February, 2010.

CCSC: Midwest, Keynote Speaker, "A New Way of Thinking about Computational Thinking", St. Xavier University, Chicago, October 2009.

National Academies: Workshop on Computational Thinking February 2009.

Problem-Centric Learning, Sept 2008, Rochester Institute of Technology

What is Computer Science?, April 2008, NSF CPATH, Living in the Knowledge Society.

Problems in AP Computer Science, June 2008, Advanced Placement Computer Science Reading, Professional Development Night.

CPATH: Science and Computer Science Purde University, November 2007.

CPATH: Problem-based Learning Keynote, Workshop sponsored by Argonne Labs and Governors University, November 2007.

Microsoft Computational Thinking Summit Redmond, WA, September 2007.

Google Faculty Summit, Mountain View, CA, July 2007.

HarambeNet: Introducing Computer Science through Modeling and Analysis of Social Networks SIGCSE 2007 Workshop, with Jeffrey Forbes.

The Cruelty of Really Teaching Computer Science Redux, University of California, Riverside, January 2006.

The Cruelty of Really Teaching Computer Science Redux, University of British Columbia Computer Science Distinguished Lecturer Series, Fall 2005.

The Cruelty of Really Teaching Computer Science Redux, University of Washington Computer Science Distinguished Lecturer Series, Fall 2005.

The Cruelty of Really Teaching Computer Science Redux, Keynote talk at CCSC/SE, Consortium of Computing Sciences in Colleges, Southeast US, November 2005.

The Cruelty of Really Teaching Computer Science Redux, Keynote talk at CCSC/E, Consortium of Computing Sciences in Colleges, Eastern US, October 2005.

A Random Walk Through Computer Science, Invited/Keynote Talk at ACM/Student conference Reflections/Projections, University Illinois, Oct. 2004.

20 Years of Teaching Computer Science, invited talk at NSF Workshop for high school teachers, Stonehill College, October 2004.

Everything I Needed to Know about Programming and Computer Science I Learned from my Teachers, Keynote Talk, SIGCSE, Norfolk 2004.

Using Patterns in the First Year, invited tutorial and presentation as part of the 2000 Eastern Small College Computing Conference, University of Scranton, PA.

OO Design and Patterns, invited speaker at NSF-sponsored workshop for high school teachers at Stonehill College, June 2000.

Advanced OO Programming, invited speaker at NSF-sponsored workshop for high school teachers at Stonehill College, January 1999.

Object-Oriented Design and Programming. Three-day lecture/workshop co-taught with David Gries delivered to educators from business colleges in Denmark, October, 1998.

Possible Futures for CS2 (panelist). *SIGCSE Technical Symposium on Computer Science Education*. Atlanta, GA, 1998.

Teaching C++ in AP Courses: Four day workshop designed and delivered for the College Board, June and August, 1997.

The First Computer Science Course and C++: Paradigm Lost or Regained. DIMACS workshop on Discrete Mathematics, July, 1996.

C++ in the Advanced Placement Program. AP Computer Science Reading, Professional Night, Clemson, SC. June, 1996.

Use of C++ for CS1 and CS2, Computing Science Conference, Philadelphia, PA, 1996.

The First Year: Beyond Language Issues, (moderator and proposer), *SIGCSE Technical Symposium on Computer Science Education*. Philadelphia, PA, 1996.

Advanced Placement and C++: Opening a Dialogue, (moderator and proposer), *SIGCSE Technical Symposium on Computer Science Education*. Philadelphia, PA, 1996.

Object-Oriented Programming: How to "Scale Up" CS1. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

Themes and Tapestries: A Diversity of Approaches to Computer Science for Liberal Arts Students. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

Using Case Studies in Computer Science Courses. *SIGCSE Technical Symposium on Computer Science Education*. Phoenix, Arizona, 1994.

On Computer Science and Teaching Computer Science with some perspectives from automated Reasoning. Bryn Mawr College, January 1993.

Faster, Fairer and More Consistent Grading Techniques: Lessons From the Advanced Placement Reading *SIGCSE Technical Symposium on Computer Science Education*. Washington D.C., 1990.

The Pleasures and Perils of Teaching Introductory Computer Science. *North Carolina Council of Teachers of Mathematics Conference*, November 1990.

Teaching Recursion in Introductory Computer Science Courses. *North Carolina Council of Teachers of Mathematics Conference*. November 1986.

IX. Professional Affiliations

Member of ACM, IEEE Computer Society, SIGPLAN, SIGCSE, SIGACT, SIGCHI, SIGART, SIGSOFT, Sigma Xi.

X. Research Funding

2014 NSF CNS Collaborative Research: CS10K: Infusing Cooperative Learning into Computer Science Principles Courses to Promote Engagement and Diversity, $104,714 (collaborative proposal with Rutgers and U. Alabama, this is Duke portion).

2014 NSF REU Supplement $8,000 to support REU student Katharine Cummings, supplement to NSF 1246919

2014 NSF Supplement $15,000 supplment to support travel for CS Principles, NSF grant 1246919

2013 NSF DRK-12 $127,151 *Enhancing Computer Science Eduction in Grades 9-12 (ECSE)*, collaborative proposal to Duke, with Twin Cities Public TV, submitted, declined.

2013 NSF DRK-12 $2.5 million *Enhancing Computer Science Eduction in Grades 90-12 (ECSE)*, main grant (PI) proposal to Twin Cities Public TV, submitted, declined.

2013 NSF REU Supplement $7,680 to support CSURF student Eleanor Mehlenbacher.

2013, NSF STEP, *Addressing Retention at its Base: Building Strong Foundations in Early Courses through Mastery Learning*, $107,021, submitted, *declined*.

2013, NSF CNS, *Collaborative Research: Broadening Participation in Computer Science: AP Computer Science Principles Phase II*, award to College Board, PI $5,210,014

2013, NSF CNS, *Collaborative Research: Broadening Participation in Computer Science: AP Computer Science Principles Phase II*, award to Duke University, $543,900

2011, NSF IIS, Special Projects: *Using Computational Thinking to Model a New Course: Advanced Placement Computer Science: Principles*, $402,441, supplemental funding request to College Board, PI.

2009, NSF IIS, Special Projects: *Using Computational Thinking to Model a New Course: Advanced Placement Computer Science: Principles*, $2,093,450.00, funding to College Board, PI.

2008, NSF BPC, Computational Thinking and Fluency in the 21^{st} Century. $98,415. Submitted by College Board, PI.

2007, NSF CPATH, CISE Distinguished Education Fellow *Interdisciplinary Problem- and Case-based Computer Science*, $250K over two years, one of two inaugural CDEF awardees in Computer Science.

2004 IBM Faculty Fellow, *Issues in Large-scale Software Componentization*, $40K grant.

2002-2003 $28K, IBM Echelon: Eclipse Help for Learning Online.

2002 $400K. IBM SUR, Education/Research Grant for establishing COD (Cluster On Demand) (co-PI with Richard Lucic, Amin Vahdat, Jeff Chase).

2002 $1,500, OOPSLA Educator's Grant: Using Design Patterns (declined)

2001 $1,500, OOPSLA Educator's Grant: Using Design Patterns (declined)

2001 $120,000 IBM Education/Research Grant for establishing a teaching/cluster classroom (co-PI with Susan Rodger, Richard Lucic, Kishor Trivedi, Amin Vahdat, Jeff Chase, the $120,000 is just the education part of the grant, each of three groups was awarded a similar amount. This was part of a $1.7 million SUR grant from IBM, some of which was software and related support.)

2000-2005, $480,826, NSF, *Modules and Courses for Ubiquitous and Mobile Computing*, NSF CRCD 0088078, PI, co-PIs Prof. Carla Ellis, Prof. Amin Vahdat.

2000, $1.19 million Microsoft *Interactive Research/Teaching Classroom*, with Jeffrey Vitter, Richard Lucic, Jeffrey Chase, Carla Ellis,Deitolf Ramm, Susan Rodger, Amin Vahdat. This includes $750,000 in software support and the rest in equipment, construction, and staffing support.

1998 $223,179 equipment *Establishing Interactive Collaborative Classrooms* Hewlett-Packard University Grants Program, co-PI with Susan Rodger

1998 $1,500, OOPSLA Educator's Grant: Using Design Patterns

1998 $50,000 Microsoft Eductional Development Grant, co-PI with Susan Rodger and Jeffrey Vitter

1998–2001 $150,306 U.S. Dept of Education GANN (co-PI with Jeff Chase, Carla Ellis, Alvin Lebeck, Jeffrey Vitter)

1997–1998 $80,000 equipment (part of a $1.6 million grant) from Intel supporting computer science education at Duke.

1997 $1,200, OOPSLA Educator's Grant: Using Design Patterns

1997–2002, $200,004, "Using and Developing Design Patterns", National Science Foundation: CAREER Program, CAREER #9702550

1996, $1,700, OOPSLA Educator's Grant: Using Design Patterns

1996–1997, $119,382, "The Applied Apprenticeship Approach (AAA): An Object-Oriented/Object-Based Framework for CS2", National Science Foundation Course and Curriculum Development, grant#DUE-9554910.

1996–2001, $607,800, "CURIOUS: The Center for Undergraduate Education and Research: Integration Through Performance and Visualization", NSF CISE Educational Infrastructure Program, grant #CISE-9634475 (co-PI with Prof. Susan Rodger)

1996–1998, $13,080 "U.S.-Germany Cooperative Research to Enhance the Performance of the Model Elimination Proof Procedure" (co-PI with Prof. Donald Loveland), National Science Foundation INT-9514375

1995, $1,000, OOPSLA Educator's Grant: Design Patterns in the Introductory CS Curriculum.

XI. Research Interests

Problem-centric Learning,Software Architecture, Object-oriented systems and languages, Computer Science Education, Networked and Distributed Computing, Automated Theorem Proving, Automated Reasoning, Parallel and Distributed Computing.

XII. Teaching

2002, Winner of the Richard K Lublin Award. Cited for "Ability to engender genuine intellectual excitement, ability to engender curiousity, knowledge of field and ability to communicate that knowledge, organizational skills, creative arrangement of course."

1999, Winner of Outstanding Instructor of Computer Science Award while on sabbatical at University of British Columbia (teaching CPSC 252, a course on Data Structures for approximately 250 Engineering students).

1995, Trinity College, Robert B. Cox Distinguished Teaching in Science Award. Cited for "knowledge of field and ability to communicate it to students, openness to students, skill in organizing courses, commitment to teaching over time."

1994, 1996, 2001, 2002 Nominated by undergraduates for outstanding teaching/faculty award (one of approximately 30 faculty nominated campus-wide each year).

Course and Curriculum Design/Implementation

CPS 53 – Program Analysis and Design I, Fall 1993

In 1993 I led the redesign of the core courses for majors (CPS 06, 08, and 100) to introduce object-oriented programming using C++ and an apprentice style of learning. This led to the development of new courses, described below. This redesign was a significant departure from the C-based courses that had been in place for three years requiring a large-scale change in philosophy as well as significant efforts in developing programming libraries to make the use of C++ feasible for students with no programming experience. This redesign has led to several publications and an NSF grant awarded in December of 1995.

CPS 100E - Program Analysis and Design II, Fall 1995 –

Created a new course for students with programming experience acquired elsewhere (not at Duke), replacing Computer Science 8 and accelerating students into the major. The course reviews material from the end of CPS 6 and then covers material from CPS 100. A formal laboratory component makes this possible. (with S. Rodger)

CPS 108 – Software Design and Implementation, Spring 1995 –

Designed and taught a new course required for all majors (in 1994 formalized the requirement). The course covers advanced object-oriented programming; introducing GUI programming using C++ and Java while emphasizing significant individual and team projects using object-oriented design, analysis, and programming.

In 1995 I established a new software design and engineering component of the curriculum via the course CPS 108. This curricular change led to an NSF CAREER grant for using and developing patterns in teaching software design and introductory programming.

CPS 149S - Problem Solving Seminar, Fall 1994

Created a new seminar course for problem solving, to prepare students for the ACM programming contest. Students worked previous contest problems once a week, and two mini-contests were held. Two teams participated in the regional contest with one team placing first.

CPS 182S – Technical and Social Analysis of Information and the Internet

Designed to meet the needs of Duke's Curriculum 2000. Satisfies, research and writing requirements and science technology and society requirement.

> The development of technical and social standards governing the Internet and Information Technology in general. The role of software as it relates to law, patents, intellectual property, and IETF (Internet Engineering Task Force) standards. Written analysis of issues from a technical perspective with an emphasis on the role of software; but also on how standards relate to social and ethical issues.

In 2002 I designed and had approved CPS 182S, *Technical and Social Analysis of Information and the Internet* a course which R, W, and STS designations as part of Duke's Curriculum 2000 (research, writing, and science, technology and society, respectively). This course led to another, non-major's version of the course in 2008, to publications, and is part of the genesis for the new NSF CS Principles project.

CPS 004G – Programming for Bioinformatics

Designed as one course in a four-course, integrative and interdisciplinary program *The Genome Revolution: Society and Science* for first year students as part of Duke's FOCUS program. The course introduces programming in the context of solving problems from bioinformatics and computational biology.

In 2006 I used this course as a foundation, with work done by Alex Hartemink in our department on a more advanced course, to help spearhead and oversee the process leading to the approval of Duke's first interdepartmental and interdisciplinary minor: Computational Biology and Bioinformatics.

Compsci 82, Technical and Social Foundations of the Internet

In 2008 I used Compsci 182S (see above) as a model for developing Compsci 82, a course without the writing component, but with an Ethical Inquiry (EI) designation. This course was taught in the fall of 2008 to 239 students, the third largest enrollment for a one-section course at Duke. In 2009 I taught this course to 345 students, the second largest course at Duke and the largest course that does not satisfy a major requirement. In 2010 I again taught the course to 345 students; the course was the largest single-section course taught at Duke.

Compsci 6, Introduction to Computer Science

In 2010 I oversaw the development of a new, introductory course in computer science: Compsci 6, the new description for the course follows.

> Introduction practices and principles of computer science and programming and their impact on and potential to change the world. Algorithmic, problem-solving, and programming techniques in domains such as art, data visualization, mathematics, natural and social sciences. Programming using high-level languages and design techniques emphasizing abstraction, encapsulation, and problem decomposition. Design, implementation, testing, and analysis of algorithms and programs. No previous programming experience required.

This course is intended to appeal to a wider and more diverse audience than our previous version of the course. I developed the infrastructure for the course including developing materials used to teach Python, deciding on the libraries used, and developing the software infrastructure to support the use of Python. I worked with Robert Duvall to deliver the first version of this course in the fall of 2010.

Compsci 92, Information and the Internet

In 2013 I piloted a new version of 82 (see above) with a strong quantitative component. The new course will be an exemplar for CS Principles. I have offered that course for three years, each spring in 2013-2015. It is now at a reasonable steady-state with excellent student evaluations, an integrated two-hour lab, both a data and a programming component, and work by students to understand ethical issues related to computing in science and society.

Courses Taught

In the list below, CPS 08/53 corresponds to CS 1 and CPS 100/103 corresponds to CS 2 (courses were renumbered in 1994). CPS 206 is a graduate level programming-languages course. CPS 10 is a comprehensive/breadth first introduction to Computer Science for non-majors.

Date		Number	Title	Enrollment
2015	Fall	Compsci 101	Introduction to Computer Science	201
2015	Spring	Compsci 92	Information and the Internet	92
		Compsci 342s	Technical and Social Foundations of the Internet	18
2014	Fall	*teaching leave*		
2014	Spring	Compsci 92	Information and the Internet	81
		Compsci 342s	Technical and Social Foundations of the Internet	18
2013	Fall	Compsci 110	Information, Society, the Internet (Bass)	42
2013	Spring	Compsci 92	Information and the Internet	71
		Compsci 342s	Technical and Social Foundations of the Internet	18
2012	Fall	Compsci 101	Introduction to Computer Science	225
		Compsci 201	(oversight course taught by others)	124
2012	Spring	Compsci 101	Introduction to Computer Science	143
		Compsci 182s	Technical and Social Foundations	18
2011	Fall	Compsci 89s	Robotics/Service Learning	12
		Compsci 100	Data Structure and Algorithms	165
		Compsci 82	Technical and Social Foundations of the Internet	346
2011	Spring	Compsci 149s	Problem Solving Seminar	8
		Compsci 006	Introduction to Computer Science	142
		Compsci 182s	Technical and Social Foundations of the Internet	18
2010	Spring	Compsci 149s	Problem Solving Seminar	12
		Compsci 100	Program Design and Analysis II	55
		Compsci 182s	Technical and Social Foundations of the Internet	18
	Fall	Compsci 149s	Problem Solving Seminar	8
		Compsci 006	Introduction to Computer Science	79
		Compsci 82	Technical and Social Foundations of the Internet	345
2009	Spring	Compsci 149s	Problem Solving Seminar	11
		Compsci 100	Program Design and Analysis II	29
		Compsci 182s	Technical and Social Foundations of the Internet	20
	Fall	Compsci 149s	Problem Solving Seminar	8
		Compsci 100	Program Design and Analysis II	41
		Compsci 82	Technical and Social Foundations of the Internet	345
2008	Spring	Compsci 149S	Problem Solving Seminar	7+
		Compsci 100	Program Design and Analysis II	33
		Compsci 82S	Technical and Social Foundations of the Internet	23
	Fall	Compsci 82	Technical and Social Foundations of the Internet 239	
		Compsci 100	Program Design and Analysis II	39
		Compsci 149s	Problem Solving Seminar	8+
2007	Spring	Compsci 149S	Problem Solving Seminar	8+
	Fall	Compsci 4G	Genomics Programming (FOCUS)	16
		Compsci 108	Software Design	44
		Compsci 149s	Problem Solving Seminar	5+
2006	Spring	CPS 182s	Technical and Social Analysis of the Internet	20
		CPS 100	Program Design and Analysis II	24
		CPS 149S	Problem Solving Seminar	4+
	Fall	CPS 004g	Introduction to Programming with Genomics (FOCUS)	11
		CPS 100	Program Design and Analysis II	36
		CPS 149S	Problem Solving Seminar	4+
2005	Fall	CPS 004G	Introduction to Programming with Genomics (FOCUS)	17
		CPS 108	Software Design and Implementation	32
		CPS 149S	Problem Solving Seminar	3
2005	Spring	CPS 182s	Technical and Social Analysis of the Internet	15
		CPS 149S	Problem Solving Seminar	4
2004	Fall	CPS 006G	Introduction to Programming with Genomics (FOCUS)	15
		CPS 100	Program Design and Analysis II	35
		CPS 149S	Problem Solving Seminar	7
2004	Spring	CPS 100	Program Design and Analysis II	35
		CPS 149S	Problem Solving Seminar	7
		CPS 108	Software Design and Implementation	45
2003	Fall	CPS 006X	Program Design and Analysis I (honors)	20

Date		Number	Title	Enrollment
2002	Fall	CPS 182s	Technical and Social Analysis of the Internet	43
		CPS 149S	Problem Solving Seminar	5
2002	Spring	CPS 100	Program Design and Analysis II	105
2001	Fall	CPS 108	Software Design and Implementation	64
		CPS 06	Program Design adn Analysis I	105
		CPS 149S	Problem Solving Seminar	13
2001	Spring	CPS 108	Software Design and Implementation	124
		CPS 100	Program Design and Analysis II	114
		CPS 189S	CS Education Seminar	3
2000	Fall	CPS 100	Program Design and Analysis II	88
		CPS 149S	Problem Solving Seminar	14
		CPS 189S	CS Education Seminar	6
2000	Spring	CPS 100	Program Design and Analysis II	107
1999	Fall	CPS 108	Software Design and Implementation	115
		CPS 06	Program Design and Analysis I	173
1998	Fall	CS 252 (UBC)	Data Structures,CS2	163
1998	Spring	CPS 108	Software Design and Implementation	94
		CPS 196	Advanced Topics in OO Technology (seminar)	14
1997	Fall	CPS 100	Program Design and Analysis II	61
		CPS 108	Software Design and Implementation	63
		CPS 149S	Problem Solving Seminar	18
1997	Spring	CPS 100	Program Design and Analysis II	72
		CPS 108	Software Design and Implementation	65
1996	Fall	CPS 100E	Program Design and Analysis II	70
		CPS 108	Software Design and Implementation	58
		CPS 149S	Problem Solving Seminar	17
1996	Spring	CPS 100	Program Design and Analysis II	72
		CPS 108	Software Design and Implementation	40
		CPS 208	Software Design and Implementation	15

Date		Number	Title	Enrollment
1995	Fall	CPS 6	Intro. to Program Design/Analysis I	121
		(Team taught with S. Rodger)		
		CPS 100	Program Design and Analysis II	56
		(Team taught with S. Rodger)		
		CPS 100E	Program Design and Analysis II	55
		(Team taught with S. Rodger)		
		CPS 149S	Problem Solving Seminar	14
		(Team taught with S. Rodger)		
	Spring	CPS 108	Software Design and Implementation	38
		CPS 8	Intro. to Program/Design Analysis I	70
1994	Fall	CPS 8	Intro. to Program Design/Analysis I	63
		CPS 100	Program Design and Analysis II	45
	Spring	CPS 8	Intro. to Program Design/Analysis I	67
		CPS 100	Program Design and Analysis II	43
1993	Fall	CPS 8	Intro. to Program Design/Analyis I	39
		CPS 100	Program Design and Analysis II	29
	Spring	CPS 100	Program Design and Analysis II	48
		CPS 206	Programming Languages (graduate)	6
1992	Spring	CPS 10	Fundamentals of Computing	67
1991	Fall	CPS 10	Fundamentals of Computing	135
1987	Fall	CPS 51	Introduction to Programming	90
1987	Spring	CPS 51	Introduction to Programming	101
1986	Fall	CPS 51	Introduction to Programming	196

Thesis Advising

2004: Megan Murphy, *The Uses of Pair Programming in Introductory Computer Science Courses*, thesis for graduation with distinction.

2004: Megan Gessner, *Generation of Spanish Verb Conjugations*, thesis for graduation with distinction.

2002: Donald Onyango, *Comparison of Eduational Tools*, Masters Thesis.

1998: Matthew Kotler, *An interactive CD-based Guide to Duke University*, undergraduate thesis for graduation with Highest Distinction.

1998: Eric Jewart, *Programming in CS 0*, undergraduate thesis for graduation with High Distinction.

1998: Tafawa Kesler, *GOODS: A Design and Class Building Tool*, undergraduate thesis for graduation with Distinction.

1997: Chih-ping Fu, *Towards a Java Bean Building and Using Environment*, Masters Thesis.

Independent Study

Fall 2006, Spring 2007 *Computer and Mathematical Models of Insulin Pathways* Tiffany Chen, graduation with distinction, interdepartmental major, Computer Science and Biology (supervised from Biology by Fred Nijhout).

Fall 2005, Industry/Academic software developement with .NET technologies (with Glaxo-Smith-Kline)

Spring 2005, Web-tools for Russian Vocabulary

Spring 2004, Agile Methods and Programming for Spanish

Fall 2002, Patterns for Networked Games

Fall 2002, Online Grading

Fall 2001, Database-backed web sites: issues and solutions.

Fall 2000, A Framework for an online Calendar System supporting IETF standards

Spring 1998, Developing a CD-based guide to Duke, Advanced OO Design: A Class Browser

Fall 1997, Advanced Object Oriented Design, Interactive Web-based Journaling, Developing a CD-based guide to Duke

Spring 1997, Graphical Debugging

Spring 1997, Distance Learning using a Java Whiteboard

Fall 1996, On-line help by harvesting information with a GUI front end.

Fall 1996, Using C++ in High School Teaching.

Fall 1995, Graphics and Game programming for the Macintosh (6 students).

Summer 1995, Using C++ in High School Teaching

Spring 1995, A GUI/OO interface for air-quality modeling.

Spring 1995, Graphics Programming for the Macintosh.

Spring 1995, Object-Oriented Programming with Smalltalk.

Fall 1994, Implementing an Online Teacher Course Evaluation Book.

Spring 1994, An application-driven approach to foundations of computer science.

Summer 1994, The role of Computer Science for secondary school mathematics teachers.

Summer 1994, Computation Structures and Machine Organization.

Fall 1989, Graphical Display and Manipulation of Data Structures for the Macintosh.

EXHIBIT B: DOCUMENTS AND INFORMATION REVIEWED

Oracle v Google
10-cv-03561-WHA
Exhibit B: Materials Considered to Opening Expert Report of Dr. Owen Astrachan Relating to Fair Use

All Trial Transcripts
All Admitted Trial Exhibits
GOOG-00000221
GOOG-00000316
GOOG-00000434
GOOG-10002049-50
GOOG-00000379-82
GOOG-00000383
GOOG-10002044-48
GOOG-00000454-57
GOOG-0000513-16
GOOG-10002075-77
GOOG-00000420-22
OAGOOGLE0016828014
OAGOOGLE0024805825
OAGOOGLE0025057076
TX 1062
TX 1077
TX 2259
TX 2259
TX 2341
TX 2906
TX 2939
Deposition of Daniel Bornstein, July 22, 2011
Deposition of Daniel Bornstein, May 16, 2011
Deposition of Daniel Bornstein, November 21, 2011
Deposition of Donald Smith, November 20, 2015
Deposition of John Duimovich, December 21, 2015
Deposition of Terrence Barr, December 9, 2015
Deposition of Mike Ringhofer, December 2, 2015
Deposition of John Duimovich, December 21, 2015 Exhibit 1410
Deposition of John Duimovich, December 21, 2015 Exhibit 1411
Deposition of John Duimovich, December 21, 2015 Exhibit 1412
DX 1368
DX 1321
Oracle's First Amended Complaint
Google's Answer to First Amended Complaint and Counterclaims
Oracle's Supplemental Responses to Google's Interrogatories, Set No. 1
Oracle's Response and Objections to Google 1st RFAs (Nos. 1-429), August 4, 2011
"Free and Open Source Java-FAQ" available at https://web.archive.org/web/20080911192443/http://www.sun.com/software/opensource/java/faq.jsp#g23
The Android developer website at android.com

Oracle v Google
10-cv-03561-WHA
Exhibit B: Materials Considered to Opening Expert Report of Dr. Owen Astrachan Relating to Fair Use

The Oracle Java websites at java.sun.com and java.oracle.com, including http://java.sun.com/docs/white/platform/javaplatform.doc1.html, http://java.sun.com/docs/books/jls/first_edition/html/index.html, http://java.sun.com/docs/glossary.html
Source code, documentation, and discussion boards and blogs for Oracle's implementation of the APIs at issue, including http://download.oracle.com/javase/5/docs/index.html, http://download.oracle.com/javase/1.4/docs/index.html, and http://markmail.org/thread/xwyxemce75vvz33h#query:+page:1+mid:vnipd7bqzs5vxfjw+state:results
Source code and documentation for Android's implementation of the APIs at issue, including http://developer.android.com/reference/packages.html and http://android.git.kernel.org/?p=platform/libcore.git;a=history;f=luni/src/test/java/org/apache/harmony/security/tests/java/security/CodeSourceTest.java;hb=a49d9caee4cd74c0d2cf83d79b8ecdc00453dff8
Source code and documentation for Apache Harmony's implementation of the APIs at issue, including http://svn.apache.org, http://harmony.apache.org/faq.html and http://harmony.apache.org/subcomponents/classlibrary/compat.html
Source code and documentation for GNU Classpath's implementation of the APIs at issue, including http://www.gnu.org/software/classpath/docs/
Wikipedia, Application programming interface, http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=437864024 (as of July 13, 2011, 00:30 GMT)
Newton's Telecom Dictionary, 25th Edition
Oracle SQL: The Essential Reference," David C. Kreines (2000)
C Reference Manual, Dennis Ritchie, 1975, available at http://www.cs.bell-labs.com/who/dmr/cman.pdf
ZLib Manual, available at http://www.zlib.net/manual.html
Merriam-Webster Dictionary online
TheFreeDictionary.com
"Revised Report on the Algorithmic Language ALGOL 68", available at http://www.fh-jena.de/~kleine/history/languages/Algol68-RevisedReport.pdf
"The C Family of Languages: Interview with Dennis Ritchie, Bjarne Stroustrup, and James Gosling," Java Report, 5(7), July 2000, available at http://www.gotw.ca/publications/c_family_interview.htm
"The Feeling of Java," James Gosling, Computer, Vol. 30, Issue 6, June 1997
Mastering Regular Expressions, Jeffrey E. F. Friedl, O'Reilly and Associates, 1997
Programming Techniques: Regular expression search algorithm," Ken Thompson, Communications of the ACM, Vol. 11, Issue 6, June 1968
"SEQUEL: A structured English query language," Proc. ACM SIGFIDET Workshop, May 1974, pp. 249-264
JDBC 3.0 Specification, available at http://jcp.org/aboutJava/communityprocess/first/jsr054/jdbc-3_0-pfd-spec.pdf
X/Open: Data Management: SQL Call Level Interface (CLI), available at http://pubs.opengroup.org/onlinepubs/009654899/toc.pdf

Oracle v Google
10-cv-03561-WHA
Exhibit B: Materials Considered to Opening Expert Report of Dr. Owen Astrachan Relating to Fair Use

Linux kernel 2.4 source code as available at http://git.kernel.org/?p=linux/kernel/git/stable/linux-2.4.37.y.git, including http://git.kernel.org/?p=linux/kernel/git/stable/linux-2.4.37.y.git;a=blob;f=fs/proc/array.c;h=335226246dcafa18864e87c2f7be68f48a50b924;hb=HEAD
Solaris source code and revision history as available at http://cvs.opensolaris.org/source/xref/onnv, including http://cvs.opensolaris.org/source/history/onnv/onnv-gate/usr/src/uts/common/syscall/uucopy.c and http://cvs.opensolaris.org/source/history/onnv/onnv-gate/usr/src/uts/common/brand/; and also as archived at http://hg.genunix.org/onnv-gate.hg/; including http://hg.genunix.org/onnv-gate.hg/rev/4c5722bc28dc
BrandZ documentation at http://hub.opensolaris.org/bin/view/Community+Group+brandz/WebHome and related web pages, including http://hub.opensolaris.org/bin/download/Community+Group+brandz/WebHome/brandzoverview.pdf and http://hub.opensolaris.org/bin/view/Community+Group+brandz/design
"Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux," 2002, available at http://kernel.org/doc/ols/2002/ols2002-pages-479-495.pdf
Futex(2) manual page, available at http://www.kernel.org/doc/man-pages/online/pages/man2/futex.2.html
"Excel functions (by category)," http://office.microsoft.com/en-us/excel-help/excel-functions-by-category-HP005204211.aspx
"Calc Functions listed by category," http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category
Visicalc Reference Card, available at http://www.bricklin.com/history/refcard1.htm
"Oracle® Database Application Developer's Guide - Fundamentals," available at http://download.oracle.com/docs/cd/B14117_01/appdev.101/b10795/toc.htm
"System R: relational approach to database management," M. M. Astrachan et al; ACM Transactions on Database Systems; Vol. 1, Issue 2, June 1976.
Schildt, Herbert, Java – The Complete Reference 3-11 (9th Ed. 2014)
Wikipedia available at http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=437864024
Wikipedia available at https://en.wikipedia.org/wiki/Application_programming_interface
Webkit https://webkit.org/licensing-webkit/
An Overview of the Android Architecture available at http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture
The Android Source code available at http://source.android.com/source/index.html
Welcome to the Android Open Source Project! available at http://source.android.com/

Oracle v Google
10-cv-03561-WHA
Exhibit B: Materials Considered to Opening Expert Report of Dr. Owen Astrachan Relating to Fair Use

Android Anatomy and Physiology available at https://sites.google.com/site/io/anatomy--physiology-of-an-android/Android-Anatomy-GoogleIO.pdf?attredirects=0
"Why Bother Open Sourcing Java?" Simon Phipps August 16, 2008
"Learn Java for Android Development" Jeff Friesen (2nd Ed. 2013)
The Java Language Specification, Third Edition
The Java Language Specification, James Gosling
GNU Classpath License available at http_www.gnu.org_software_classpath_license.html
GNU Classpath Hacker's Guide available at http://www.gnu.org/software/classpath/docs/cp-hacking.html
Futex(2) manual page, available at http://www.kernel.org/doc/man-pages/online/pages/man2/futex.2.html
OpenJDK Mobile Project available at http://openjdk.java.net/projects/mobile/
OpenJDK Android Platform Implementation Details available at http://openjdk.java.net/projects/mobile/android.html
"The code is coming! The code is coming! available at http://mail.openjdk.java.net/pipermail/jdk6-dev/2008-February/000001.html
OpenJDK available at http://openjdk.java.net/
OpenJDK/jdk6/jdk6 available at http_hg.openjdk.java.net_jdk6_jdk6
J2SE 5.0 available at http://www.oracle.com/technetwork/java/javase/index-jsp-135232.html
Java Programming solving problems with software available at https://www.coursera.org/learn/java-programming
Your First Cup: An Introduction to the Java EE Platform available at http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf
Your First Cup available at http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html
Java Software available at https://www.oracle.com/java/index.html
Glossary available at http://www.oracle.com/technetwork/java/glossary-135216.html
The Oxford Dictionaries Word of the Year 2013 available at http://blog.oxforddictionaries.com/press-releases/oxford-dictionaries-word-of-the-year-2013/
Licenses available at https://source.android.com/source/licenses.html
Java SE at a Glance available at http://www.oracle.com/technetwork/java/javase/overview/index.html
Package java.lang available at http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/package-summary.html
Oracle Java SE Embedded: Developer's Guide available at https://docs.oracle.com/javase/8/embedded/develop-apps-platforms/overview.htm
The Java Application Programming Interface, Volume 1: Core Packages
Codenames, Tags, and Build Numbers available at https://source.android.com/source/build-numbers.html
The Java Language Specification, First Edition
net.lang.c available at http://groups.google.com/group/net.lang.c/browse_thread/thread/6409987225e13a31/50da7fdd143184bd?q=regex#50da7fdd143184bd

EXHIBIT C: EXCEL AND STAROFFICE SPREADSHEET FUNCTION NAMES

Microsoft Excel 2003	Oracle Open Office Calc (Today)
ABS	ABS
ACCRINT	ACCRINT
ACCRINTM	ACCRINTM
ACOS	ACOS
ACOSH	ACOSH
	ACOT
	ACOTH
ADDRESS	ADDRESS
AMORDEGRC	AMORDEGRC
AMORLINC	AMORLINC
AND	AND
	ARABIC
AREAS	AREAS
ASC	
ASIN	ASIN
ASINH	ASINH
ATAN	ATAN
ATAN2	ATAN2
ATANH	ATANH
AVEDEV	AVEDEV
AVERAGE	AVERAGE
AVERAGEA	AVERAGEA
BAHTTEXT	BAHTTEXT
	BASE
BESSELI	BESSELI
BESSELJ	BESSELJ
BESSELK	BESSELK
BESSELY	BESSELY
BETADIST	BETADIST
BETAINV	BETAINV
BIN2DEC	BIN2DEC
BIN2HEX	BIN2HEX
BIN2OCT	BIN2OCT

BINOMDIST	BINOMDIST
CEILING	CEILING
CELL	CELL
CHAR	CHAR
CHIDIST	CHIDIST
CHIINV	CHIINV
	CHISQDIST
	CHISQINV
CHITEST	CHITEST
CHOOSE	CHOOSE
CLEAN	CLEAN
CODE	CODE
COLUMN	COLUMN
COLUMNS	COLUMNS
COM	
COMBIN	COMBIN
	COMBINA
COMPLEX	COMPLEX
CONCATENATE	CONCATENATE
CONFIDENCE	CONFIDENCE
CONVERT	CONVERT
	CONVERT_ADD
CORREL	CORREL
COS	COS
COSH	COSH
	COT
	COTH
COUNT	COUNT
COUNTA	COUNTA
COUNTBLANK	COUNTBLANK
COUNTIF	COUNTIF
COUPDAYBS	COUPDAYBS
COUPDAYS	COUPDAYS
COUPDAYSNC	COUPDAYSNC
COUPNCD	COUPNCD
COUPNUM	COUPNUM
COUPPCD	COUPPCD

COVAR	COVAR
CRITBINOM	CRITBINOM
CUMIPMT	CUMIPMT
	CUMIPMT_ADD
CUMPRINC	CUMPRINC
	CUMPRINC_ADD
	CURRENT
DATE	DATE
DATEVALUE	DATEVALUE
DAVERAGE	DAVERAGE
DAY	DAY
	DAYS
DAYS360	DAYS360
	DAYSINMONTH
	DAYSINYEAR
DB	DB
DCOUNT	DCOUNT
DCOUNTA	DCOUNTA
DDB	DDB
	DDE
DEC2BIN	DEC2BIN
DEC2HEX	DEC2HEX
DEC2OCT	DEC2OCT
	DECIMAL
DEGREES	DEGREES
DELTA	DELTA
DEVSQ	DEVSQ
DGET	DGET
DISC	DISC
DMAX	DMAX
DMIN	DMIN
DOLLAR	DOLLAR
DOLLARDE	DOLLARDE
DOLLARFR	DOLLARFR
DPRODUCT	DPRODUCT
DSTDEV	DSTDEV
DSTDEVP	DSTDEVP

DSUM	DSUM
DURATION	DURATION
	DURATION_ADD
DVAR	DVAR
DVARP	DVARP
	EASTERSUNDAY
EDATE	EDATE
EFFECT	EFFECT_ADD
	EFFECTIVE
EOMONTH	EOMONTH
ERF	ERF
ERFC	ERFC
ERROR.TYPE	ERRORTYPE
EUROCONVERT	
EVEN	EVEN
EXACT	EXACT
EXP	EXP
EXPONDIST	EXPONDIST
FACT	FACT
FACTDOUBLE	FACTDOUBLE
FALSE	FALSE
FDIST	FDIST
FIND	FIND
FINDB	
FINV	FINV
FISHER	FISHER
FISHERINV	FISHERINV
FIXED	FIXED
FLOOR	FLOOR
FORECAST	FORECAST
	FORMULA
FREQUENCY	FREQUENCY
FTEST	FTEST
FV	FV
FVSCHEDULE	FVSCHEDULE
	GAMMA
GAMMADIST	GAMMADIST

GAMMAINV	GAMMAINV
GAMMALN	GAMMALN
	GAUSS
GCD	GCD
	GCD_ADD
GEOMEAN	GEOMEAN
GESTEP	GESTEP
GETPIVOTDATA	
GROWTH	GROWTH
HARMEAN	HARMEAN
HEX2BIN	HEX2BIN
HEX2DEC	HEX2DEC
HEX2OCT	HEX2OCT
HLOOKUP	HLOOKUP
HOUR	HOUR
HYPERLINK	HYPERLINK
HYPGEOMDIST	HYPGEOMDIST
IF	IF
IMABS	IMABS
IMAGINARY	IMAGINARY
IMARGUMENT	IMARGUMENT
IMCONJUGATE	IMCONJUGATE
IMCOS	IMCOS
IMDIV	IMDIV
IMEXP	IMEXP
IMLN	IMLN
IMLOG10	IMLOG10
IMLOG2	IMLOG2
IMPOWER	IMPOWER
IMPRODUCT	IMPRODUCT
IMREAL	IMREAL
IMSIN	IMSIN
IMSQRT	IMSQRT
IMSUB	IMSUB
IMSUM	IMSUM
INDEX	INDEX
INDIRECT	INDIRECT

INFO	INFO
INT	INT
INTERCEPT	INTERCEPT
INTRATE	INTRATE
IPMT	IPMT
IRR	IRR
ISBLANK	ISBLANK
ISERR	ISERR
ISERROR	ISERROR
ISEVEN	ISEVEN
	ISEVEN_ADD
	ISFORMULA
	ISLEAPYEAR
ISLOGICAL	ISLOGICAL
ISNA	ISNA
ISNONTEXT	ISNONTEXT
ISNUMBER	ISNUMBER
ISODD	ISODD
	ISODD_ADD
ISPMT	ISPMT
ISREF	ISREF
ISTEXT	ISTEXT
IT	
JIS	
KURT	KURT
LARGE	LARGE
LCM	LCM
	LCM_ADD
LEFT	LEFT
LEFTB	
LEN	LEN
LENB	
LINEST	LINEST
LN	LN
LOG	LOG
LOG10	LOG10
LOGEST	LOGEST

LOGINV	LOGINV
LOGNORMDIST	LOGNORMDIST
LOOKUP	LOOKUP
LOWER	LOWER
MATCH	MATCH
MAX	MAX
MAXA	MAXA
MDETERM	MDETERM
MDURATION	MDURATION
MEDIAN	MEDIAN
MID	MID
MIDB	
MIN	MIN
MINA	MINA
MINUTE	MINUTE
MINVERSE	MINVERSE
MIRR	MIRR
MMULT	MMULT
MOD	MOD
MODE	MODE
MONTH	MONTH
	MONTHS
MROUND	MROUND
MULTINOMIAL	MULTINOMIAL
	MUNIT
NA	NA
NEGBINOMDIST	NEGBINOMDIST
NETWORKDAYS	NETWORKDAYS
NOMINAL	NOMINAL
	NOMINAL_ADD
NORMDIST	NORMDIST
NORMINV	NORMINV
NORMSDIST	NORMSDIST
NORMSINV	NORMSINV
NOT	NOT
NOW	NOW
NPER	NPER

NPV	NPV
OCT2BIN	OCT2BIN
OCT2DEC	OCT2DEC
OCT2HEX	OCT2HEX
ODD	ODD
ODDFPRICE	ODDFPRICE
ODDFYIELD	ODDFYIELD
ODDLPRICE	ODDLPRICE
ODDLYIELD	ODDLYIELD
OFFSET	OFFSET
OR	OR
PEARSON	PEARSON
PERCENTILE	PERCENTILE
PERCENTRANK	PERCENTRANK
PERMUT	PERMUT
PHONETIC	
	PERMUTATIONA
	PHI
PI	PI
PMT	PMT
POISSON	POISSON
POWER	POWER
PPMT	PPMT
PRICE	PRICE
PRICEDISC	PRICEDISC
PRICEMAT	PRICEMAT
PROB	PROB
PRODUCT	PRODUCT
PROPER	PROPER
PV	PV
QUARTILE	QUARTILE
QUOTIENT	QUOTIENT
RADIANS	RADIANS
RAND	RAND
RANDBETWEEN	RANDBETWEEN
RANK	RANK
RATE	RATE

RECEIVED	RECEIVED
REPLACE	REPLACE
REPLACEB	
REPT	REPT
RIGHT	RIGHT
RIGHTB	
ROMAN	ROMAN
ROUND	ROUND
ROUNDDOWN	ROUNDDOWN
ROUNDUP	ROUNDUP
ROW	ROW
ROWS	ROWS
	RRI
RSQ	RSQ
RTD	
SEARCH	SEARCH
SEARCHB	
SECOND	SECOND
SERIESSUM	SERIESSUM
	SHEET
	SHEETS
SIGN	SIGN
SIN	SIN
SINH	SINH
SKEW	SKEW
SLN	SLN
SLOPE	SLOPE
SMALL	SMALL
SQL.REQUEST	
SQRT	SQRT
SQRTPI	SQRTPI
STANDARDIZE	STANDARDIZE
STDEV	STDEV
STDEVA	STDEVA
STDEVP	STDEVP
STDEVPA	STDEVPA
STEYX	STEYX

	STYLE
SUBSTITUTE	SUBSTITUTE
SUBTOTAL	SUBTOTAL
SUM	SUM
SUMIF	SUMIF
SUMPRODUCT	SUMPRODUCT
SUMSQ	SUMSQ
SUMX2MY2	SUMX2MY2
SUMX2PY2	SUMX2PY2
SUMXMY2	SUMXMY2
SYD	SYD
TAN	TAN
TANH	TANH
TBILLEQ	TBILLEQ
TBILLPRICE	TBILLPRICE
TBILLYIELD	TBILLYIELD
TDIST	TDIST
TEXT	TEXT
TIME	TIME
TIMEVALUE	TIMEVALUE
TINV	TINV
TODAY	TODAY
TRANSPOSE	TRANSPOSE
TREND	TREND
TRIM	TRIM
TRIMMEAN	TRIMMEAN
TRUE	TRUE
TRUNC	TRUNC
TTEST	TTEST
TYPE	TYPE
UPPER	UPPER
VALUE	VALUE
VAR	VAR
VARA	VARA
VARP	VARP
VARPA	VARPA
VDB	VDB

VLOOKUP	VLOOKUP
WEEKDAY	WEEKDAY
WEEKNUM	WEEKNUM
	WEEKNUM_ADD
	WEEKS
	WEEKSINYEAR
WEIBULL	WEIBULL
WORKDAY	WORKDAY
XIRR	XIRR
XNPV	XNPV
YEAR	YEAR
YEARFRAC	YEARFRAC
	YEARS
YIELD	YIELD
YIELDDISC	YIELDDISC
YIELDMAT	YIELDMAT
ZTEST	ZTEST

EXHIBIT D: LX_BRAND SYSCALL TABLE

From lx_brand/common/lx_brand.c:

```
static struct lx_sysent sysents[] = {

        {"nosys",       NULL,           NOSYS_NULL,     0},     /*  0 */

        {"exit",        lx_exit,        0,              1},     /*  1 */

        {"fork",        lx_fork,        0,              0},     /*  2 */

        {"read",        lx_read,        0,              3},     /*  3 */

        {"write",       write,          SYS_PASSTHRU,   3},     /*  4 */

        {"open",        lx_open,        0,              3},     /*  5 */

        {"close",       close,          SYS_PASSTHRU,   1},     /*  6 */

        {"waitpid",     lx_waitpid,     0,              3},     /*  7 */

        {"creat",       creat,          SYS_PASSTHRU,   2},     /*  8 */

        {"link",        lx_link,        0,              2},     /*  9 */

        {"unlink",      lx_unlink,      0,              1},     /* 10 */

        {"execve",      lx_execve,      0,              3},     /* 11 */

        {"chdir",       chdir,          SYS_PASSTHRU,   1},     /* 12 */

        {"time",        lx_time,        0,              1},     /* 13 */

        {"mknod",       lx_mknod,       0,              3},     /* 14 */

        {"chmod",       lx_chmod,       0,              2},     /* 15 */

        {"lchown16",    lx_lchown16,    0,              3},     /* 16 */

        {"break",       NULL,           NOSYS_OBSOLETE, 0},     /* 17 */

        {"stat",        NULL,           NOSYS_OBSOLETE, 0},     /* 18 */

        {"lseek",       lx_lseek,       0,              3},     /* 19 */

        {"getpid",      lx_getpid,      0,              0},     /* 20 */

        {"mount",       lx_mount,       0,              5},     /* 21 */
```

```
{"umount",      lx_umount,      0,              1},     /* 22 */
{"setuid16",    lx_setuid16,    0,              1},     /* 23 */
{"getuid16",    lx_getuid16,    0,              0},     /* 24 */
{"stime",       stime,          SYS_PASSTHRU,   1},     /* 25 */
{"ptrace",      lx_ptrace,      0,              4},     /* 26 */
{"alarm",       (int (*)())alarm, SYS_PASSTHRU, 1},     /* 27 */
{"fstat",       NULL,           NOSYS_OBSOLETE, 0},     /* 28 */
{"pause",       pause,          SYS_PASSTHRU,   0},     /* 29 */
{"utime",       lx_utime,       0,              2},     /* 30 */
{"stty",        NULL,           NOSYS_OBSOLETE, 0},     /* 31 */
{"gtty",        NULL,           NOSYS_OBSOLETE, 0},     /* 32 */
{"access",      access,         SYS_PASSTHRU,   2},     /* 33 */
{"nice",        nice,           SYS_PASSTHRU,   1},     /* 34 */
{"ftime",       NULL,           NOSYS_OBSOLETE, 0},     /* 35 */
{"sync",        lx_sync,        0,              0},     /* 36 */
{"kill",        lx_kill,        0,              2},     /* 37 */
{"rename",      lx_rename,      0,              2},     /* 38 */
{"mkdir",       mkdir,          SYS_PASSTHRU,   2},     /* 39 */
{"rmdir",       lx_rmdir,       0,              1},     /* 40 */
{"dup",         dup,            SYS_PASSTHRU,   1},     /* 41 */
{"pipe",        lx_pipe,        0,              1},     /* 42 */
{"times",       lx_times,       0,              1},     /* 43 */
{"prof",        NULL,           NOSYS_OBSOLETE, 0},     /* 44 */
{"brk",         lx_brk,         0,              1},     /* 45 */
{"setgid16",    lx_setgid16,    0,              1},     /* 46 */
{"getgid16",    lx_getgid16,    0,              0},     /* 47 */
```

```
{"signal",      lx_signal,      0,              2},     /* 48 */
{"geteuid16",   lx_geteuid16,   0,              0},     /* 49 */
{"getegid16",   lx_getegid16,   0,              0},     /* 50 */
{"acct",        NULL,           NOSYS_NO_EQUIV, 0},     /* 51 */
{"umount2",     lx_umount2,     0,              2},     /* 52 */
{"lock",        NULL,           NOSYS_OBSOLETE, 0},     /* 53 */
{"ioctl",       lx_ioctl,       0,              3},     /* 54 */
{"fcntl",       lx_fcntl,       0,              3},     /* 55 */
{"mpx",         NULL,           NOSYS_OBSOLETE, 0},     /* 56 */
{"setpgid",     lx_setpgid,     0,              2},     /* 57 */
{"ulimit",      NULL,           NOSYS_OBSOLETE, 0},     /* 58 */
{"olduname",    NULL,           NOSYS_OBSOLETE, 0},     /* 59 */
{"umask",       (int (*)())umask, SYS_PASSTHRU, 1},     /* 60 */
{"chroot",      chroot,         SYS_PASSTHRU,   1},     /* 61 */
{"ustat",       lx_ustat,       0,              2},     /* 62 */
{"dup2",        lx_dup2,        0,              2},     /* 63 */
{"getppid",     lx_getppid,     0,              0},     /* 64 */
{"getpgrp",     lx_getpgrp,     0,              0},     /* 65 */
{"setsid",      lx_setsid,      0,              0},     /* 66 */
{"sigaction",   lx_sigaction,   0,              3},     /* 67 */
{"sgetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 68 */
{"ssetmask",    NULL,           NOSYS_OBSOLETE, 0},     /* 69 */
{"setreuid16",  lx_setreuid16,  0,              2},     /* 70 */
{"setregid16",  lx_setregid16,  0,              2},     /* 71 */
{"sigsuspend",  lx_sigsuspend,  0,              1},     /* 72 */
{"sigpending",  lx_sigpending,  0,              1},     /* 73 */
```

```
{"sethostname", lx_sethostname, 0,              2},     /* 74 */
{"setrlimit",   lx_setrlimit,   0,              2},     /* 75 */
{"getrlimit",   lx_oldgetrlimit, 0,             2},     /* 76 */
{"getrusage",   lx_getrusage,   0,              2},     /* 77 */
{"gettimeofday", lx_gettimeofday, 0,            2},     /* 78 */
{"settimeofday", lx_settimeofday, 0,            2},     /* 79 */
{"getgroups16", lx_getgroups16, 0,              2},     /* 80 */
{"setgroups16", lx_setgroups16, 0,              2},     /* 81 */
{"select",      NULL,           NOSYS_OBSOLETE, 0},     /* 82 */
{"symlink",     symlink,        SYS_PASSTHRU,   2},     /* 83 */
{"oldlstat",    NULL,           NOSYS_OBSOLETE, 0},     /* 84 */
{"readlink",    readlink,       SYS_PASSTHRU,   3},     /* 85 */
{"uselib",      NULL,           NOSYS_KERNEL,   0},     /* 86 */
{"swapon",      NULL,           NOSYS_KERNEL,   0},     /* 87 */
{"reboot",      lx_reboot,      0,              4},     /* 88 */
{"readdir",     lx_readdir,     0,              3},     /* 89 */
{"mmap",        lx_mmap,        0,              6},     /* 90 */
{"munmap",      munmap,         SYS_PASSTHRU,   2},     /* 91 */
{"truncate",    lx_truncate,    0,              2},     /* 92 */
{"ftruncate",   lx_ftruncate,   0,              2},     /* 93 */
{"fchmod",      fchmod,         SYS_PASSTHRU,   2},     /* 94 */
{"fchown16",    lx_fchown16,    0,              3},     /* 95 */
{"getpriority", lx_getpriority, 0,              2},     /* 96 */
{"setpriority", lx_setpriority, 0,              3},     /* 97 */
{"profil",      NULL,           NOSYS_NO_EQUIV, 0},     /* 98 */
{"statfs",      lx_statfs,      0,              2},     /* 99 */
```

```
{"fstatfs",     lx_fstatfs,     0,              2},     /* 100 */
{"ioperm",      NULL,           NOSYS_NO_EQUIV, 0},     /* 101 */
{"socketcall",  lx_socketcall,  0,              2},     /* 102 */
{"syslog",      NULL,           NOSYS_KERNEL,   0},     /* 103 */
{"setitimer",   lx_setitimer,   0,              3},     /* 104 */
{"getitimer",   getitimer,      SYS_PASSTHRU,   2},     /* 105 */
{"stat",        lx_stat,        0,              2},     /* 106 */
{"lstat",       lx_lstat,       0,              2},     /* 107 */
{"fstat",       lx_fstat,       0,              2},     /* 108 */
{"uname",       NULL,           NOSYS_OBSOLETE, 0},     /* 109 */
{"oldiopl",     NULL,           NOSYS_NO_EQUIV, 0},     /* 110 */
{"vhangup",     lx_vhangup,     0,              0},     /* 111 */
{"idle",        NULL,           NOSYS_NO_EQUIV, 0},     /* 112 */
{"vm86old",     NULL,           NOSYS_OBSOLETE, 0},     /* 113 */
{"wait4",       lx_wait4,       0,              4},     /* 114 */
{"swapoff",     NULL,           NOSYS_KERNEL,   0},     /* 115 */
{"sysinfo",     lx_sysinfo,     0,              1},     /* 116 */
{"ipc",         lx_ipc,         0,              5},     /* 117 */
{"fsync",       lx_fsync,       0,              1},     /* 118 */
{"sigreturn",   lx_sigreturn,   0,              1},     /* 119 */
{"clone",       lx_clone,       0,              5},     /* 120 */
{"setdomainname", lx_setdomainname, 0,          2},     /* 121 */
{"uname",       lx_uname,       0,              1},     /* 122 */
{"modify_ldt",  lx_modify_ldt,  0,              3},     /* 123 */
{"adjtimex",    lx_adjtimex,    0,              1},     /* 124 */
{"mprotect",    lx_mprotect,    0,              3},     /* 125 */
```

```
{"sigprocmask", lx_sigprocmask, 0,              3},     /* 126 */
{"create_module", NULL,         NOSYS_KERNEL,   0},     /* 127 */
{"init_module", NULL,           NOSYS_KERNEL,   0},     /* 128 */
{"delete_module", NULL,         NOSYS_KERNEL,   0},     /* 129 */
{"get_kernel_syms", NULL,       NOSYS_KERNEL,   0},     /* 130 */
{"quotactl",    NULL,           NOSYS_KERNEL,   0},     /* 131 */
{"getpgid",     lx_getpgid,     0,              1},     /* 132 */
{"fchdir",      fchdir,         SYS_PASSTHRU,   1},     /* 133 */
{"bdflush",     NULL,           NOSYS_KERNEL,   0},     /* 134 */
{"sysfs",       lx_sysfs,       0,              3},     /* 135 */
{"personality", lx_personality, 0,              1},     /* 136 */
{"afs_syscall", NULL,           NOSYS_KERNEL,   0},     /* 137 */
{"setfsuid16",  lx_setfsuid16,  0,              1},     /* 138 */
{"setfsgid16",  lx_setfsgid16,  0,              1},     /* 139 */
{"llseek",      lx_llseek,      0,              5},     /* 140 */
{"getdents",    getdents,       SYS_PASSTHRU,   3},     /* 141 */
{"select",      lx_select,      0,              5},     /* 142 */
{"flock",       lx_flock,       0,              2},     /* 143 */
{"msync",       lx_msync,       0,              3},     /* 144 */
{"readv",       lx_readv,       0,              3},     /* 145 */
{"writev",      lx_writev,      0,              3},     /* 146 */
{"getsid",      lx_getsid,      0,              1},     /* 147 */
{"fdatasync",   lx_fdatasync,   0,              1},     /* 148 */
{"sysctl",      lx_sysctl,      0,              1},     /* 149 */
{"mlock",       lx_mlock,       0,              2},     /* 150 */
{"munlock",     lx_munlock,     0,              2},     /* 151 */
```

```
        {"mlockall",    lx_mlockall,    0,              1},     /* 152 */
        {"munlockall",  lx_munlockall,  0,              0},     /* 153 */
        {"sched_setparam", lx_sched_setparam,   0,      2},     /* 154 */
        {"sched_getparam", lx_sched_getparam,   0,      2},     /* 155 */
        {"sched_setscheduler", lx_sched_setscheduler, 0, 3},    /* 156 */
        {"sched_getscheduler", lx_sched_getscheduler, 0, 1},    /* 157 */
        {"sched_yield", (int (*)())yield, SYS_PASSTHRU, 0},     /* 158 */
        {"sched_get_priority_max", lx_sched_get_priority_max, 0, 1}, /* 159
*/
        {"sched_get_priority_min", lx_sched_get_priority_min, 0, 1}, /* 160
*/
        {"sched_rr_get_interval", lx_sched_rr_get_interval, 0,  2},  /* 161
*/
        {"nanosleep",   nanosleep,      SYS_PASSTHRU,   2},     /* 162 */
        {"mremap",      NULL,           NOSYS_NO_EQUIV, 0},     /* 163 */
        {"setresuid16", lx_setresuid16, 0,              3},     /* 164 */
        {"getresuid16", lx_getresuid16, 0,              3},     /* 165 */
        {"vm86",        NULL,           NOSYS_NO_EQUIV, 0},     /* 166 */
        {"query_module", lx_query_module, NOSYS_KERNEL, 5},     /* 167 */
        {"poll",        lx_poll,        0,              3},     /* 168 */
        {"nfsservctl",  NULL,           NOSYS_KERNEL,   0},     /* 169 */
        {"setresgid16", lx_setresgid16, 0,              3},     /* 170 */
        {"getresgid16", lx_getresgid16, 0,              3},     /* 171 */
        {"prctl",       NULL,           NOSYS_UNDOC,    0},     /* 172 */
        {"rt_sigreturn", lx_rt_sigreturn, 0,            0},     /* 173 */
        {"rt_sigaction", lx_rt_sigaction, 0,            4},     /* 174 */
```

```
{"rt_sigprocmask", lx_rt_sigprocmask, 0,         4},     /* 175 */
{"rt_sigpending", lx_rt_sigpending, 0,           2},     /* 176 */
{"rt_sigtimedwait", lx_rt_sigtimedwait, 0,       4},     /* 177 */
{"sigqueueinfo", NULL,           NOSYS_UNDOC,    0},     /* 178 */
{"rt_sigsuspend", lx_rt_sigsuspend, 0,           2},     /* 179 */
{"pread64",     lx_pread64,      0,              5},     /* 180 */
{"pwrite64",    lx_pwrite64,     0,              5},     /* 181 */
{"chown16",     lx_chown16,      0,              3},     /* 182 */
{"getcwd",      lx_getcwd,       0,              2},     /* 183 */
{"capget",      NULL,            NOSYS_NO_EQUIV, 0},     /* 184 */
{"capset",      NULL,            NOSYS_NO_EQUIV, 0},     /* 185 */
{"sigaltstack", lx_sigaltstack, 0,               2},     /* 186 */
{"sendfile",    lx_sendfile,     0,              4},     /* 187 */
{"getpmsg",     NULL,            NOSYS_OBSOLETE, 0},     /* 188 */
{"putpmsg",     NULL,            NOSYS_OBSOLETE, 0},     /* 189 */
{"vfork",       lx_vfork,        0,              0},     /* 190 */
{"getrlimit",   lx_getrlimit,    0,              2},     /* 191 */
{"mmap2",       lx_mmap2,        EBP_HAS_ARG6,   6},     /* 192 */
{"truncate64",  lx_truncate64,   0,              3},     /* 193 */
{"ftruncate64", lx_ftruncate64, 0,               3},     /* 194 */
{"stat64",      lx_stat64,       0,              2},     /* 195 */
{"lstat64",     lx_lstat64,      0,              2},     /* 196 */
{"fstat64",     lx_fstat64,      0,              2},     /* 197 */
{"lchown",      lchown,          SYS_PASSTHRU,   3},     /* 198 */
{"getuid",      (int (*)())getuid, SYS_PASSTHRU, 0},     /* 199 */
{"getgid",      (int (*)())getgid, SYS_PASSTHRU, 0},     /* 200 */
```

```
{"geteuid",     lx_geteuid,     0,              0},     /* 201 */
{"getegid",     lx_getegid,     0,              0},     /* 202 */
{"setreuid",    setreuid,       SYS_PASSTHRU,   0},     /* 203 */
{"setregid",    setregid,       SYS_PASSTHRU,   0},     /* 204 */
{"getgroups",   getgroups,      SYS_PASSTHRU,   2},     /* 205 */
{"setgroups",   lx_setgroups,   0,              2},     /* 206 */
{"fchown",      lx_fchown,      0,              3},     /* 207 */
{"setresuid",   lx_setresuid,   0,              3},     /* 208 */
{"getresuid",   lx_getresuid,   0,              3},     /* 209 */
{"setresgid",   lx_setresgid,   0,              3},     /* 210 */
{"getresgid",   lx_getresgid,   0,              3},     /* 211 */
{"chown",       lx_chown,       0,              3},     /* 212 */
{"setuid",      setuid,         SYS_PASSTHRU,   1},     /* 213 */
{"setgid",      setgid,         SYS_PASSTHRU,   1},     /* 214 */
{"setfsuid",    lx_setfsuid,    0,              1},     /* 215 */
{"setfsgid",    lx_setfsgid,    0,              1},     /* 216 */
{"pivot_root",  NULL,           NOSYS_KERNEL,   0},     /* 217 */
{"mincore",     mincore,        SYS_PASSTHRU,   3},     /* 218 */
{"madvise",     lx_madvise,     0,              3},     /* 219 */
{"getdents64",  lx_getdents64,  0,              3},     /* 220 */
{"fcntl64",     lx_fcntl64,     0,              3},     /* 221 */
{"tux",         NULL,           NOSYS_NO_EQUIV, 0},     /* 222 */
{"security",    NULL,           NOSYS_NO_EQUIV, 0},     /* 223 */
{"gettid",      lx_gettid,      0,              0},     /* 224 */
{"readahead",   NULL,           NOSYS_NO_EQUIV, 0},     /* 225 */
{"setxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 226 */
```

```
{"lsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 227 */
{"fsetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 228 */
{"getxattr",    NULL,           NOSYS_NO_EQUIV, 0},     /* 229 */
{"lgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 230 */
{"fgetxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 231 */
{"listxattr",   NULL,           NOSYS_NO_EQUIV, 0},     /* 232 */
{"llistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 233 */
{"flistxattr",  NULL,           NOSYS_NO_EQUIV, 0},     /* 234 */
{"removexattr", NULL,           NOSYS_NO_EQUIV, 0},     /* 235 */
{"lremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 236 */
{"fremovexattr", NULL,          NOSYS_NO_EQUIV, 0},     /* 237 */
{"tkill",       lx_tkill,       0,              2},     /* 238 */
{"sendfile64",  lx_sendfile64,  0,              4},     /* 239 */
{"futex",       lx_futex,       EBP_HAS_ARG6,   6},     /* 240 */
{"sched_setaffinity",   lx_sched_setaffinity,   0, 3},  /* 241 */
{"sched_getaffinity",   lx_sched_getaffinity,   0, 3},  /* 242 */
{"set_thread_area", lx_set_thread_area, 0,      1},     /* 243 */
{"get_thread_area", lx_get_thread_area, 0,      1},     /* 244 */
{"io_setup",    NULL,           NOSYS_NO_EQUIV, 0},     /* 245 */
{"io_destroy",  NULL,           NOSYS_NO_EQUIV, 0},     /* 246 */
{"io_getevents", NULL,          NOSYS_NO_EQUIV, 0},     /* 247 */
{"io_submit",   NULL,           NOSYS_NO_EQUIV, 0},     /* 248 */
{"io_cancel",   NULL,           NOSYS_NO_EQUIV, 0},     /* 249 */
{"fadvise64",   NULL,           NOSYS_UNDOC,    0},     /* 250 */
{"nosys",       NULL,           0,              0},     /* 251 */
{"group_exit",  lx_group_exit,  0,              1},     /* 252 */
```

```
{"lookup_dcookie", NULL,        NOSYS_NO_EQUIV, 0},     /* 253 */
{"epoll_create", NULL,          NOSYS_NO_EQUIV, 0},     /* 254 */
{"epoll_ctl",   NULL,           NOSYS_NO_EQUIV, 0},     /* 255 */
{"epoll_wait",  NULL,           NOSYS_NO_EQUIV, 0},     /* 256 */
{"remap_file_pages", NULL,      NOSYS_NO_EQUIV, 0},     /* 257 */
{"set_tid_address", lx_set_tid_address, 0,      1},     /* 258 */
{"timer_create", NULL,          NOSYS_UNDOC,    0},     /* 259 */
{"timer_settime", NULL,         NOSYS_UNDOC,    0},     /* 260 */
{"timer_gettime", NULL,         NOSYS_UNDOC,    0},     /* 261 */
{"timer_getoverrun", NULL,      NOSYS_UNDOC,    0},     /* 262 */
{"timer_delete", NULL,          NOSYS_UNDOC,    0},     /* 263 */
{"clock_settime", lx_clock_settime,     0,      2},     /* 264 */
{"clock_gettime", lx_clock_gettime,     0,      2},     /* 265 */
{"clock_getres", lx_clock_getres,       0,      2},     /* 266 */
{"clock_nanosleep", lx_clock_nanosleep, 0,      4},     /* 267 */
{"statfs64",    lx_statfs64,    0,              2},     /* 268 */
{"fstatfs64",   lx_fstatfs64,   0,              2},     /* 269 */
{"tgkill",      lx_tgkill,      0,              3},     /* 270 */
/* The following system calls only exist in kernel 2.6 and greater */
{"utimes",      utimes,         SYS_PASSTHRU,   2},     /* 271 */
{"fadvise64_64", NULL,          NOSYS_NULL,     0},     /* 272 */
{"vserver",     NULL,           NOSYS_NULL,     0},     /* 273 */
{"mbind",       NULL,           NOSYS_NULL,     0},     /* 274 */
{"get_mempolicy", NULL,         NOSYS_NULL,     0},     /* 275 */
{"set_mempolicy", NULL,         NOSYS_NULL,     0},     /* 276 */
{"mq_open",     NULL,           NOSYS_NULL,     0},     /* 277 */
```

```
{"mq_unlink",   NULL,           NOSYS_NULL,     0},     /* 278 */
{"mq_timedsend", NULL,          NOSYS_NULL,     0},     /* 279 *
{"mq_timedreceive", NULL,       NOSYS_NULL,     0},     /* 280 */
{"mq_notify",   NULL,           NOSYS_NULL,     0},     /* 281 */
{"mq_getsetattr", NULL,         NOSYS_NULL,     0},     /* 282 */
{"kexec_load",  NULL,           NOSYS_NULL,     0},     /* 283 */
{"waitid",      lx_waitid,      0,              4},     /* 284 */
{"sys_setaltroot", NULL,        NOSYS_NULL,     0},     /* 285 */
{"add_key",     NULL,           NOSYS_NULL,     0},     /* 286 */
{"request_key", NULL,           NOSYS_NULL,     0},     /* 287 */
{"keyctl",      NULL,           NOSYS_NULL,     0},     /* 288 */
{"ioprio_set",  NULL,           NOSYS_NULL,     0},     /* 289 */
{"ioprio_get",  NULL,           NOSYS_NULL,     0},     /* 290 */
{"inotify_init", NULL,          NOSYS_NULL,     0},     /* 291 */
{"inotify_add_watch", NULL,     NOSYS_NULL,     0},     /* 292 */
{"inotify_rm_watch", NULL,      NOSYS_NULL,     0},     /* 293 */
{"migrate_pages", NULL,         NOSYS_NULL,     0},     /* 294 */
{"openat",      lx_openat,      0,              4},     /* 295 */
{"mkdirat",     lx_mkdirat,     0,              3},     /* 296 */
{"mknodat",     lx_mknodat,     0,              4},     /* 297 */
{"fchownat",    lx_fchownat,    0,              5},     /* 298 */
{"futimesat",   lx_futimesat,   0,              3},     /* 299 */
{"fstatat64",   lx_fstatat64,   0,              4},     /* 300 */
{"unlinkat",    lx_unlinkat,    0,              3},     /* 301 */
{"renameat",    lx_renameat,    0,              4},     /* 302 */
{"linkat",      lx_linkat,      0,              5},     /* 303 */
```

```
        {"symlinkat",    lx_symlinkat,    0,              3},     /* 304 */
        {"readlinkat",   lx_readlinkat,   0,              4},     /* 305 */
        {"fchmodat",     lx_fchmodat,     0,              4},     /* 306 */
        {"faccessat",    lx_faccessat,    0,              4},     /* 307 */
        {"pselect6",     NULL,            NOSYS_NULL,     0},     /* 308 */
        {"ppoll",        NULL,            NOSYS_NULL,     0},     /* 309 */
        {"unshare",      NULL,            NOSYS_NULL,     0},     /* 310 */
        {"set_robust_list", NULL,         NOSYS_NULL,     0},     /* 311 */
        {"get_robust_list", NULL,         NOSYS_NULL,     0},     /* 312 */
        {"splice",       NULL,            NOSYS_NULL,     0},     /* 313 */
        {"sync_file_range", NULL,         NOSYS_NULL,     0},     /* 314 */
        {"tee",          NULL,            NOSYS_NULL,     0},     /* 315 */
        {"vmsplice",     NULL,            NOSYS_NULL,     0},     /* 316 */
        {"move_pages",   NULL,            NOSYS_NULL,     0},     /* 317 */
};
```

EXHIBIT E: SOURCE CODE FOR SLOCCOUNTER.PY AND SLOCCOUNTERTOTAL.PY

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections,re

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aprivdict = {}
aset = set()
amethnames = []
apubclass = set()

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jprivdict = {}
jset = set()
jmethnames = []
jpubclass = set()

gcdict = collections.defaultdict(int)
gperclass = collections.defaultdict(int)
gprivdict = {}
gset = set()
gmethnames = []
gpubclass = set()

afunclist = []
jfunclist = []
gfunclist = []

methnames = []

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def is_func(line):
    if "new " in line:
        return False
    parts = line.split()
    if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    return False


def getClass(path):
    '''
    path ends with .java, return class name preceding .java including preceding .
    e.g., for java/lang/Arrays, return .Arrays
    '''
    nm = path[:-5]
    index = nm.rfind("/")
    return "."+nm[index+1:]

def pubtrack(fname,pubclass,clname):
    f = open(fname)
    allText = f.read()
    changedText = re.sub(r"\s+"," ",allText)
    contents = changedText.split()
    for i in range(len(contents)-2):
        if contents[i] == "public" and contents[i+1] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "interface":
```



```
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "abstract" and contents[i+2] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "final" and contents[i+2] == "class":
            pubclass.add(clname)
            break

def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames,pubclass):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True


    class_name = getClass(onepath)
    #comment out line below 12/26, no difference
    #pubtrack(onepath,pubclass,packname+class_name)
    fullname = packname+class_name
    if not fullname in pubclass:
        print "rejected",fullname
        return True

    f = open(onepath)

    pcount = 0
    first = True
    public = False
    pubf = 0
    privf = 0
    for line in f:

        line = line.strip()

        if is_func(line):
            methnames.append(line)
            if line.startswith("public"):
                pubf += 1
            else:
                privf += 1
                nm = packname+class_name
                if not nm in privdict:
                    privdict[nm] = []
                privdict[nm].append(line)

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1

                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
```

```
                    cdict["extends"] += 1
                break

        f.close()

        perclass[pcount] += 1
        if pcount == 0:
            #print "%s = %d" % (onepath,pcount)
            pass

        funclist.append((pubf,privf))
        return public


    def pop_one(packname,onepath,pubclass):

        if not onepath.endswith(".java"):
            return True
        if onepath.endswith("package-info.java"):
            return True

        class_name = getClass(onepath)
        pubtrack(onepath,pubclass,packname+class_name)

    def populate(basepath,packname,pubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = pop_one(packname,top_path,pubclass)


    def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames,p
    ubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,m
    ethnames,pubclass)
                if not c:
                    #print "no public",top_path,top
                    pass
                #print "%s has %d public" % (top_path,c)

    def func_stats(coll):
        low = 0
        word_total = 0
        wt_count = 0
        nonlow = 0
        getter = 0
        setter = 0
        req = 0
        equ = 0

        obj_names = ["toString", "hashCode", "notifyAll", "getClass", "equals","clone","wait","fin
    alize","notify"]

        for nm in coll:

            if nm in obj_names:
                req += 1
                if nm.startswith("eq"):
```



```
                    equ += 1
            elif nm.islower():
                low += 1
                #print "\t lower",nm
            else:
                wc = 0
                for i,ch in enumerate(nm):
                    if ch.isupper() and i > 0 and nm[i-1].islower():
                        wc += 1

                wc += 1
                #word_total += wc
                nonlow += 1

                if nm.startswith("get"):
                    getter += 1
                elif nm.startswith("set"):
                    setter += 1
                else:
                    word_total += wc
                    wt_count += 1

        ftot = low+nonlow+req
        print "total = %d, req = %d one = %d more = %d\n" % (ftot,req,low,nonlow)
        print "perc = %f avg = %f\n" % (1.0*low/(low+nonlow+1),1.0*word_total/(wt_count+1
    ))
        print "non simple = %d\n" % (wt_count)

        print "getter = %d, setter = %d, req = %d, equal = %d, total = %d\n" % (getter,setter,req,equ,r
    eq+getter+setter)

    def funcalyze(methnames):
        all_names = set()
        names = []
        for meth in methnames:
            if meth.startswith("public"):
                nameEnd = meth.find("(")
                if nameEnd == -1:
                    print "error on ",meth
                else:
                    name = meth[:nameEnd]
                    space = name.rfind(" ")
                    mname = name[space+1:]
                    all_names.add(mname)
                    names.append(mname)

        print "total = %d, unique = %d\n" % (len(names), len(all_names))
        print "unique"
        func_stats(all_names)
        print "total"
        func_stats(names)

        meth_counts = [(names.count(nm),nm) for nm in all_names]
        smc = sorted(meth_counts, reverse=True)
        print "top func occurrences"
        for pair in smc[:20]:
            print pair


        return all_names



    def report(cdict,perclass,funclist,privdict,methnames):

        uset = funcalyze(methnames)


        ctotal = 0
        for key in cdict:
            if key.find("public") < 0:
                continue
            print "%s occurrences = %d" % (key,cdict[key])
```

```
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "public class/interface total = %d" % (ctotal)

    ctotal = 0
    for key in cdict:
        if key.find("protected") < 0:
            continue
        print "%s occurrences = %d" % (key,cdict[key])
        if key.find("class") >= 0 or key.find("interface") >= 0:
            ctotal += cdict[key]
    print "----"
    print "protected class/interface total = %d" % (ctotal)

    print "per class method counts"
    print "# methods\t#classes"
    total = 0
    levels = collections.defaultdict(int)
    levlist = [0,1,6,11,16,21,51,101,100001]
    for method_count in sorted(perclass.keys()):
        print "%d\t%d" % (method_count,perclass[method_count])
        total += method_count*perclass[method_count]
        for lev in xrange(1,len(levlist)):
            if levlist[lev-1] <= method_count < levlist[lev]:
                levels[lev] += perclass[method_count]
    print "-----"
    print "total methods = %d" % (total)
    print "\n---summary--"
    total = 0
    for lev in xrange(1,len(levlist)):
        print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[le
v])
        total += levels[lev]
    print "total = %d" % (total)

    print "size of funclist = %d" % (len(funclist))
    total = 0
    totalMeths = 0
    totalPriv = 0
    total2 = 0
    for x in funclist:
        totalMeths += x[0] + x[1]
        totalPriv += x[1]
        if x[0] != 0 or x[1] != 0:
            total += 100.0*x[0]/(x[1]+x[0])
            total2 += 100.0*x[1]/(x[1]+x[0])
    print "average = %f" % (total/(len(funclist)+1))
    print "total meths = %d" % (totalMeths)
    print "total private = %d" % (totalPriv)
    print "averagePRIV = %f" % (total2/(len(funclist)+1))
    return uset

def analyze():
    rootpath = "/Volumes/My Passport for Mac/expert/google"
    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
    apath7 = "/Users/ola/expert/google/newsource/libcore/luni/src/main/java/"
    #apath = "/Users/ola/expert/google/S21"
    #javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    #javapath = "/Users/ola/expert/google/JDK/jdk/src/share/classes"
    #jdkpath = "/Users/ola/expert/google/java6/SUNWj6dev/reloc/jdk/instances/jdk
1.6.0"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                #"java.math",
                "java.net",
```



```
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                #"javax.xml",
                #"javax.xml.datatype",
                #"javax.xml.namespace",
                #"javax.xml.parsers",
                #"javax.xml.transform",
                #"javax.xml.transform.dom",
                #"javax.xml.transform.sax",
                #"javax.xml.transform.stream",
                #"javax.xml.validation",
                #"javax.xml.xpath"
                ]

    for pack in packages:
        populate(rootpath+apath,pack,jpubclass)
        populate(rootpath+apath7,pack,apubclass)

    allinter = jpubclass & apubclass
    print "js = %d, as = %d, gs = %d, inter = %d\n" % (len(jpubclass),len(apubclass),len(gpu
bclass),len(allinter))

    #return

    for pack in packages:
        print "android"
        topcount(rootpath+apath,pack,jcdict,jperclass,jset,jfunclist,jprivdict,j
methnames,jpubclass)
        print "android 7"
        topcount(rootpath+apath7,pack,acdict,aperclass,aset,afunclist,aprivdict,
amethnames,apubclass)


    print "%d packages analyzed" % (len(packages))
    print "\nAndroid Analysis"
    juset = report(jcdict,jperclass,jfunclist,jprivdict,jmethnames)
    print "\nAndroid7 Analysis"
    auset = report(acdict,aperclass,afunclist,aprivdict,amethnames)
    print "\n-----"

    jmset = juset
    amset = auset
    inter = jmset&amset
```

```
        aonly = amset-jmset
        jonly = jmset-amset
        print "android only count = ",len(aonly),len(amset)
        print "android 7 only count = ",len(jonly),len(jmset)
        print "android only"
        for i,n in enumerate(sorted(aonly)):
            print i,n
        print "android7 only"
        for i,n in enumerate(sorted(jonly)):
            print i,n

        # public classes that are different?
        japub = jpubclass - apubclass
        ajpub = apubclass - jpubclass
        print "androidpub = %d, android7 pub = %d, j-a = %d, a-j = %d\n" % (len(jpubclass),len(apub
    class), len(japub),len(ajpub))
        print "android public not in adroid7"
        for nm in sorted(japub):
            print nm
        print "-----\n"
        print "android7 public not in android"
        for nm in sorted(ajpub):
            print nm
        print "-----\n"


        privlog = open("androidprivatelog","w")
        for pack in aprivdict:
            if pack in jprivdict:
                line = "package class private {0!s}\n".format(pack)
                print "package class private %s" % (pack)
                privlog.write(line)
                for priv in aprivdict[pack]:
                    line = "\topenJDK {0!s}\n".format(priv)
                    privlog.write(line)
                    #print "\topenJDK %s" % (priv)
                    if priv in jprivdict[pack]:
                        privlog.write("\t\talso in Java\n")
                        #print "\t\talso in Java"
                for priv in jprivdict[pack]:
                    if not priv in aprivdict[pack]:
                        privlog.write("\tJava "+priv+"\n")
                        #print "\tJava %s" % (priv)
        privlog.close()




        print "common package/private"
        inter = jset&aset
        for name in inter:
            print name

        print "\nopenJDK\n------"
        for name in aset:
            print name
        print "\nJava\n------"
        for name in jset:
            print name



if __name__ == "__main__":
    analyze()
```

```
    '''
    Created as part of work on expert report
    for Google/Oracle for GreenbergTraurig

    @author: ola
    @copyright: owen astrachan, compsciconsulting
    '''
    import os,collections,re

    acdict = collections.defaultdict(int)
    aperclass = collections.defaultdict(int)
    aprivdict = {}
    aset = set()
    amethnames = []
    apubclass = set()

    jcdict = collections.defaultdict(int)
    jperclass = collections.defaultdict(int)
    jprivdict = {}
    jset = set()
    jmethnames = []
    jpubclass = set()

    gcdict = collections.defaultdict(int)
    gperclass = collections.defaultdict(int)
    gprivdict = {}
    gset = set()
    gmethnames = []
    gpubclass = set()

    afunclist = []
    jfunclist = []
    gfunclist = []

    methnames = []

    public_ids = ["public class",
                  "public abstract class",
                  "public interface",
                  "protected class",
                  "protected",
                  "public"]

    def is_func(line):
        if "new " in line:
            return False
        parts = line.split()
        if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
            return True
        if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
            return True
        return False


    def getClass(path):
        '''
      path ends with .java, return class name preceding .java including preceding .
      e.g., for java/lang/Arrays, return .Arrays
      '''
        nm = path[:-5]
        index = nm.rfind("/")
        return "."+nm[index+1:]

    def pubtrack(fname,pubclass,clname):
        f = open(fname)
        allText = f.read()
        changedText = re.sub(r"\s+"," ",allText)
        contents = changedText.split()
        for i in range(len(contents)-2):
            if contents[i] == "public" and contents[i+1] == "class":
                pubclass.add(clname)
                break
            if contents[i] == "public" and contents[i+1] == "interface":
```



```
                pubclass.add(clname)
                break
            if contents[i] == "public" and contents[i+1] == "abstract" and contents[i+2]
    == "class":
                pubclass.add(clname)
                break
            if contents[i] == "public" and contents[i+1] == "final" and contents[i+2] =
    = "class":
                pubclass.add(clname)
                break

    def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames,pubc
    lass):

        if not onepath.endswith(".java"):
            return True
        if onepath.endswith("package-info.java"):
            return True


        class_name = getClass(onepath)
        #comment out line below 12/26, no difference
        #pubtrack(onepath,pubclass,packname+class_name)
        fullname = packname+class_name
        if not fullname in pubclass:
            print "WOULD BE rejected",fullname

        f = open(onepath)

        pcount = 0
        first = True
        public = False
        pubf = 0
        privf = 0
        for line in f:

            line = line.strip()
            cpath = line.find("Classpath")
            if cpath >= 0:
                print "PASS",fullname,"**",line
                return True
        print "\tFAIL",fullname
        funclist.append((pubf,privf))
        return public


    def pop_one(packname,onepath,pubclass):

        if not onepath.endswith(".java"):
            return True
        if onepath.endswith("package-info.java"):
            return True

        class_name = getClass(onepath)
        pubtrack(onepath,pubclass,packname+class_name)

    def populate(basepath,packname,pubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = pop_one(packname,top_path,pubclass)


    def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames,p
    ubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
```

```
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,m
ethnames,pubclass)
                if not c:
                    #print "no public",top_path,top
                    pass
                #print "%s has %d public" % (top_path,c)

def analyze():
    rootpath = "/Volumes/My Passport for Mac/expert/google"
    javapath = "/Users/ola/expert/google/JDK/jdk/src/share/classes"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                #"java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                #"javax.xml",
                #"javax.xml.datatype",
                #"javax.xml.namespace",
                #"javax.xml.parsers",
                #"javax.xml.transform",
                #"javax.xml.transform.dom",
                #"javax.xml.transform.sax",
                #"javax.xml.transform.stream",
                #"javax.xml.validation",
                #"javax.xml.xpath"
                ]

        for pack in packages:
            populate(rootpath+javapath,pack,jpubclass)
```



```

        for pack in packages:
            print "java"
            topcount(rootpath+javapath,pack,jcdict,jperclass,jset,jfunclist,jprivdic
t,jmethnames,jpubclass)
        print "%d packages analyzed" % (len(packages))


if __name__ == "__main__":
    analyze()
```

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections,re

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aprivdict = {}
aset = set()
amethnames = []
apubclass = set()

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jprivdict = {}
jset = set()
jmethnames = []
jpubclass = set()

gcdict = collections.defaultdict(int)
gperclass = collections.defaultdict(int)
gprivdict = {}
gset = set()
gmethnames = []
gpubclass = set()

afunclist = []
jfunclist = []
gfunclist = []

methnames = []

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def is_func(line):
    if "new " in line:
        return False
    parts = line.split()
    if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    return False


def getClass(path):
    '''
    path ends with .java, return class name preceding .java including preceding .
    e.g., for java/lang/Arrays, return .Arrays
    '''
    nm = path[:-5]
    index = nm.rfind("/")
    return "."+nm[index+1:]

def pubtrack(fname,pubclass,clname):
    f = open(fname)
    allText = f.read()
    changedText = re.sub(r"\s+"," ",allText)
    contents = changedText.split()
    for i in range(len(contents)-2):
        if contents[i] == "public" and contents[i+1] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "interface":
```



```
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "abstract" and contents[i+2] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "final" and contents[i+2] == "class":
            pubclass.add(clname)
            break

def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames,pubclass):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True


    class_name = getClass(onepath)
    #comment out line below 12/26, no difference
    #pubtrack(onepath,pubclass,packname+class_name)
    fullname = packname+class_name
    if not fullname in pubclass:
        print "rejected",fullname
        return True

    f = open(onepath)

    pcount = 0
    first = True
    public = False
    pubf = 0
    privf = 0
    for line in f:

        line = line.strip()

        if is_func(line):
            methnames.append(line)
            if line.startswith("public"):
                pubf += 1
            else:
                privf += 1
                nm = packname+class_name
                if not nm in privdict:
                    privdict[nm] = []
                privdict[nm].append(line)

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1

                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
```

```
                    cdict["extends"] += 1
                break

    f.close()

    perclass[pcount] += 1
    if pcount == 0:
        #print "%s = %d" % (onepath,pcount)
        pass

    funclist.append((pubf,privf))
    return public


def pop_one(packname,onepath,pubclass):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True

    class_name = getClass(onepath)
    pubtrack(onepath,pubclass,packname+class_name)

def populate(basepath,packname,pubclass):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    if not os.path.exists(packagepath):
        print "*** error no package",packname
        return
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = pop_one(packname,top_path,pubclass)


def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames,pubclass):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    if not os.path.exists(packagepath):
        print "*** error no package",packname
        return
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,methnames,pubclass)
            if not c:
                #print "no public",top_path,top
                pass
            #print "%s has %d public" % (top_path,c)

def func_stats(coll):
    low = 0
    word_total = 0
    wt_count = 0
    nonlow = 0
    getter = 0
    setter = 0
    req = 0
    equ = 0

    obj_names = ["toString", "hashCode", "notifyAll", "getClass", "equals","clone","wait","finalize","notify"]
```



```
    for nm in coll:

        if nm in obj_names:
            req += 1
            if nm.startswith("eq"):
                equ += 1
        elif nm.islower():
            low += 1
            #print "\t lower",nm
        else:
            wc = 0
            for i,ch in enumerate(nm):
                if ch.isupper() and i > 0 and nm[i-1].islower():
                    wc += 1

            wc += 1
            #word_total += wc
            nonlow += 1

            if nm.startswith("get"):
                getter += 1
            elif nm.startswith("set"):
                setter += 1
            else:
                word_total += wc
                wt_count += 1

    ftot = low+nonlow+req
    print "total = %d, req = %d one = %d more = %d\n" % (ftot,req,low,nonlow)
    print "perc = %f avg = %f\n" % (1.0*low/(low+nonlow+1),1.0*word_total/(wt_count+1))
    print "non simple = %d\n" % (wt_count)

    print "getter = %d, setter = %d, req = %d, equal = %d, total = %d\n" % (getter,setter,req,equ,req+getter+setter)

def funcalyze(methnames):
    all_names = set()
    names = []
    for meth in methnames:
        if meth.startswith("public"):
            nameEnd = meth.find("(")
            if nameEnd == -1:
                print "error on ",meth
            else:
                name = meth[:nameEnd]
                space = name.rfind(" ")
                mname = name[space+1:]
                all_names.add(mname)
                names.append(mname)

    print "total = %d, unique = %d\n" % (len(names), len(all_names))
    print "unique"
    func_stats(all_names)
    print "total"
    func_stats(names)

    meth_counts = [(names.count(nm),nm) for nm in all_names]
    smc = sorted(meth_counts, reverse=True)
    print "top func occurrences"
    for pair in smc[:20]:
        print pair


    return all_names


def report(cdict,perclass,funclist,privdict,methnames):

    uset = funcalyze(methnames)
```

```
285
        ctotal = 0
        for key in cdict:
            if key.find("public") < 0:
                continue
290         print "%s occurrences = %d" % (key,cdict[key])
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "public class/interface total = %d" % (ctotal)
295
        ctotal = 0
        for key in cdict:
            if key.find("protected") < 0:
                continue
300         print "%s occurrences = %d" % (key,cdict[key])
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "protected class/interface total = %d" % (ctotal)
305
        print "per class method counts"
        print "# methods\t#classes"
        total = 0
        levels = collections.defaultdict(int)
310     levlist = [0,1,6,11,16,21,51,101,100001]
        for method_count in sorted(perclass.keys()):
            print "%d\t%d" % (method_count,perclass[method_count])
            total += method_count*perclass[method_count]
            for lev in xrange(1,len(levlist)):
315             if levlist[lev-1] <= method_count < levlist[lev]:
                    levels[lev] += perclass[method_count]
        print "-----"
        print "total methods = %d" % (total)
        print "\n---summary---"
320     total = 0
        for lev in xrange(1,len(levlist)):
            print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[le
    v])
            total += levels[lev]
        print "total = %d" % (total)
325
        print "size of funclist = %d" % (len(funclist))
        total = 0
        totalMeths = 0
        totalPriv = 0
330     total2 = 0
        for x in funclist:
            totalMeths += x[0] + x[1]
            totalPriv += x[1]
            if x[0] != 0 or x[1] != 0:
335             total += 100.0*x[0]/(x[1]+x[0])
                total2 += 100.0*x[1]/(x[1]+x[0])
        print "average = %f" % (total/(len(funclist)+1))
        print "total meths = %d" % (totalMeths)
        print "total private = %d" % (totalPriv)
340     print "averagePRIV = %f" % (total2/(len(funclist)+1))
        return uset

    def analyze():
        rootpath = "/Volumes/My Passport for Mac/expert/google"
345     #apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
        #apath = "/Users/ola/expert/google/newsource/libcore/luni/src/main/java/"
        #apath = "/Users/ola/expert/google/S21"
        javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
        jdkpath = "/Users/ola/expert/google/JDK/jdk/src/share/classes"
350     #javapath = "/Users/ola/expert/google/java6/SUNWj6dev/reloc/jdk/instances/jd
    k1.6.0"

        print "javapath",javapath
        print "jdkpath",jdkpath

355     packages = ["java.awt.font",
```



```
                    "java.beans",
                    "java.io",
                    "java.lang",
                    "java.lang.annotation",
360                 "java.lang.ref",
                    "java.lang.reflect",
                    #"java.math",
                    "java.net",
                    "java.nio",
365                 "java.nio.channels",
                    "java.nio.channels.spi",
                    "java.nio.charset",
                    "java.nio.charset.spi",
                    "java.security",
370                 "java.security.acl",
                    "java.security.cert",
                    "java.security.interfaces",
                    "java.security.spec",
                    "java.sql",
375                 "java.text",
                    "java.util",
                    #"java.util.concurrent",
                    #"java.util.concurrrent.atomic",
                    #"java.util.concurrent.locks",
380                 "java.util.jar",
                    "java.util.logging",
                    "java.util.prefs",
                    "java.util.regex",
                    "java.util.zip",
385                 "javax.crypto",
                    "javax.crypto.interfaces",
                    "javax.crypto.spec",
                    "javax.net",
                    "javax.net.ssl",
390                 "javax.security.auth",
                    "javax.security.auth.callback",
                    "javax.security.auth.login",
                    "javax.security.auth.x500",
                    "javax.security.cert",
395                 "javax.sql",
                    #"javax.xml",
                    #"javax.xml.datatype",
                    #"javax.xml.namespace",
                    #"javax.xml.parsers",
400                 #"javax.xml.transform",
                    #"javax.xml.transform.dom",
                    #"javax.xml.transform.sax",
                    #"javax.xml.transform.stream",
                    #"javax.xml.validation",
405                 #"javax.xml.xpath"
                    ]

        for pack in packages:
            populate(rootpath+javapath,pack,jpubclass)
410         populate(rootpath+jdkpath,pack,apubclass)

        allinter = jpubclass & apubclass
        print "js = %d, as = %d, gs = %d, inter = %d\n" % (len(jpubclass),len(apubclass),len(gpu
    bclass),len(allinter))

415     #return

        for pack in packages:
            print "java"
            topcount(rootpath+javapath,pack,jcdict,jperclass,jset,jfunclist,jprivdic
    t,jmethnames,jpubclass)
420         print "jdk"
            topcount(rootpath+jdkpath,pack,acdict,aperclass,aset,afunclist,aprivdict
    ,amethnames,apubclass)


        print "%d packages analyzed" % (len(packages))
425     print "\nJava Analysis"
```

```
        juset = report(jcdict,jperclass,jfunclist,jprivdict,jmethnames)
        print "\nOpenJDK Analysis"
        auset = report(acdict,aperclass,afunclist,aprivdict,amethnames)
        print "\n------"

        jmset = juset
        amset = auset
        inter = jmset&amset
        aonly = amset-jmset
        jonly = jmset-amset
        print "jdk only count = ",len(aonly),len(amset)
        print "java only count = ",len(jonly),len(jmset)
        print "jdk only"
        for i,n in enumerate(sorted(aonly)):
            print i,n
        print "java only"
        for i,n in enumerate(sorted(jonly)):
            print i,n

        # public classes that are different?
        japub = jpubclass - apubclass
        ajpub = apubclass - jpubclass
        print "javapub = %d, android pub = %d, j-a = %d, a-j = %d\n" % (len(jpubclass),len(apubcla
    ss), len(japub),len(ajpub))
        print "java public not in openJDK"
        for nm in sorted(japub):
            print nm
        print "------\n"
        print "openJDK public not in java"
        for nm in sorted(ajpub):
            print nm
        print "------\n"


        privlog = open("JDKprivatelog","w")
        for pack in aprivdict:
            if pack in jprivdict:
                line = "package class private {0!s}\n".format(pack)
                print "package class private %s" % (pack)
                privlog.write(line)
                for priv in aprivdict[pack]:
                    line = "\topenJDK {0!s}\n".format(priv)
                    privlog.write(line)
                    #print "\topenJDK %s" % (priv)
                    if priv in jprivdict[pack]:
                        privlog.write("\t\talso in Java\n")
                        #print "\t\talso in Java"
                for priv in jprivdict[pack]:
                    if not priv in aprivdict[pack]:
                        privlog.write("\tJava "+priv+"\n")
                        #print "\tJava %s" % (priv)
        privlog.close()




        print "common package/private"
        inter = jset&aset
        for name in inter:
            print name

        print "\nopenJDK\n-------"
        for name in aset:
            print name
        print "\nJava\n-------"
        for name in jset:
            print name
```



```
    if __name__ == "__main__":
        analyze()
```

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections,re

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aprivdict = {}
aset = set()
amethnames = []
apubclass = set()

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jprivdict = {}
jset = set()
jmethnames = []
jpubclass = set()

gcdict = collections.defaultdict(int)
gperclass = collections.defaultdict(int)
gprivdict = {}
gset = set()
gmethnames = []
gpubclass = set()

afunclist = []
jfunclist = []
gfunclist = []

methnames = []

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def is_func(line):
    if "new " in line:
        return False
    parts = line.split()
    if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    return False


def getClass(path):
    '''
    path ends with .java, return class name preceding .java including preceding .
    e.g., for java/lang/Arrays, return .Arrays
    '''
    nm = path[:-5]
    index = nm.rfind("/")
    return "."+nm[index+1:]

def pubtrack(fname,pubclass,clname):
    f = open(fname)
    allText = f.read()
    changedText = re.sub(r"\s+"," ",allText)
    contents = changedText.split()
    for i in range(len(contents)-2):
        if contents[i] == "public" and contents[i+1] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "interface":
```



```
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "abstract" and contents[i+2] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "final" and contents[i+2] == "class":
            pubclass.add(clname)
            break

def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames,pubclass):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True


    class_name = getClass(onepath)
    #pubtrack(onepath,pubclass,packname+class_name)
    fullname = packname+class_name
    if not fullname in pubclass:
        print "rejected",fullname
        return True

    f = open(onepath)

    pcount = 0
    first = True
    public = False
    pubf = 0
    privf = 0
    for line in f:

        line = line.strip()

        if is_func(line):
            methnames.append(line)
            if line.startswith("public"):
                pubf += 1
            else:
                privf += 1
                nm = packname+class_name
                if not nm in privdict:
                    privdict[nm] = []
                privdict[nm].append(line)

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1

                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
```

```
                    break

        f.close()

        perclass[pcount] += 1
        if pcount == 0:
            #print "%s = %d" % (onepath,pcount)
            pass

        funclist.append((pubf,privf))
        return public


    def pop_one(packname,onepath,pubclass):

        if not onepath.endswith(".java"):
            return True
        if onepath.endswith("package-info.java"):
            return True

        class_name = getClass(onepath)
        pubtrack(onepath,pubclass,packname+class_name)

    def populate(basepath,packname,pubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = pop_one(packname,top_path,pubclass)


    def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames,pubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,methnames,pubclass)
                if not c:
                    #print "no public",top_path,top
                    pass
                #print "%s has %d public" % (top_path,c)

    def func_stats(coll):
        low = 0
        word_total = 0
        wt_count = 0
        nonlow = 0
        getter = 0
        setter = 0
        req = 0

        obj_names = ["toString", "hashCode", "notifyAll", "getClass"]

        for nm in coll:
            if nm.islower():
                low += 1
                #print "\t lower",nm
            else:
                wc = 0
                for i,ch in enumerate(nm):
                    if ch.isupper() and i > 0 and nm[i-1].islower():
```



```
                        wc += 1

                wc += 1
                #word_total += wc
                nonlow += 1

                if nm.startswith("get"):
                    getter += 1
                elif nm.startswith("set"):
                    setter += 1
                elif nm in obj_names:
                    req += 1
                else:
                    word_total += wc
                    wt_count += 1

        print "total = %d, one = %d more = %d\n" % (nonlow+low,low,nonlow)
        print "perc = %f avg = %f\n" % (1.0*low/(low+nonlow),1.0*word_total/wt_count)
        print "non simple = %d\n" % (wt_count)

        print "getter = %d, setter = %d, req = %d, total = %d\n" % (getter,setter,req,req+getter+setter)

    def funcalyze(methnames):
        all_names = set()
        names = []
        for meth in methnames:
            if meth.startswith("public"):
                nameEnd = meth.find("(")
                if nameEnd == -1:
                    print "error on ",meth
                else:
                    name = meth[:nameEnd]
                    space = name.rfind(" ")
                    mname = name[space+1:]
                    all_names.add(mname)
                    names.append(mname)

        print "total = %d, unique = %d\n" % (len(names), len(all_names))
        print "unique"
        func_stats(all_names)
        print "total"
        func_stats(names)

        meth_counts = [(names.count(nm),nm) for nm in all_names]
        smc = sorted(meth_counts, reverse=True)
        print "top func occurrences"
        for pair in smc[:20]:
            print pair


        return all_names



    def report(cdict,perclass,funclist,privdict,methnames):

        uset = funcalyze(methnames)


        ctotal = 0
        for key in cdict:
            if key.find("public") < 0:
                continue
            print "%s occurrences = %d" % (key,cdict[key])
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "-----"
        print "public class/interface total = %d" % (ctotal)

        ctotal = 0
        for key in cdict:
            if key.find("protected") < 0:
```

```
                continue
            print "%s occurrences = %d" % (key,cdict[key])
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "protected class/interface total = %d" % (ctotal)

        print "per class method counts"
        print "# methods\t#classes"
        total = 0
        levels = collections.defaultdict(int)
        levlist = [0,1,6,11,16,21,51,101,100001]
        for method_count in sorted(perclass.keys()):
            print "%d\t%d" % (method_count,perclass[method_count])
            total += method_count*perclass[method_count]
            for lev in xrange(1,len(levlist)):
                if levlist[lev-1] <= method_count < levlist[lev]:
                    levels[lev] += perclass[method_count]
        print "-----"
        print "total methods = %d" % (total)
        print "\n---summary--"
        total = 0
        for lev in xrange(1,len(levlist)):
            print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[le
v])
            total += levels[lev]
        print "total = %d" % (total)

        print "size of funclist = %d" % (len(funclist))
        total = 0
        totalMeths = 0
        totalPriv = 0
        total2 = 0
        for x in funclist:
            totalMeths += x[0] + x[1]
            totalPriv += x[1]
            if x[0] != 0 or x[1] != 0:
                total += 100.0*x[0]/(x[1]+x[0])
                total2 += 100.0*x[1]/(x[1]+x[0])
        print "average = %f" % (total/len(funclist))
        print "total meths = %d" % (totalMeths)
        print "total private = %d" % (totalPriv)
        print "averagePRIV = %f" % (total2/len(funclist))
        return uset

    def get_method(st):
        paren = st.find("(")
        sp = st.rfind(" ",0,paren)
        name = st[sp+1:paren]
        return name

    def analyze():
        rootpath = "/Volumes/My Passport for Mac/expert/google"
        apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
        javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
        gnupath = "/Users/ola/expert/google/all/gnu"

        packages = ["java.awt.font",
                    "java.beans",
                    "java.io",
                    "java.lang",
                    "java.lang.annotation",
                    "java.lang.ref",
                    "java.lang.reflect",
                    #"java.math",
                    "java.net",
                    "java.nio",
                    "java.nio.channels",
                    "java.nio.channels.spi",
                    "java.nio.charset",
                    "java.nio.charset.spi",
                    "java.security",
                    "java.security.acl",
```



```
                    "java.security.cert",
                    "java.security.interfaces",
                    "java.security.spec",
                    "java.sql",
                    "java.text",
                    "java.util",
                    #"java.util.concurrent",
                    #"java.util.concurrrent.atomic",
                    #"java.util.concurrent.locks",
                    "java.util.jar",
                    "java.util.logging",
                    "java.util.prefs",
                    "java.util.regex",
                    "java.util.zip",
                    "javax.crypto",
                    "javax.crypto.interfaces",
                    "javax.crypto.spec",
                    "javax.net",
                    "javax.net.ssl",
                    "javax.security.auth",
                    "javax.security.auth.callback",
                    "javax.security.auth.login",
                    "javax.security.auth.x500",
                    "javax.security.cert",
                    "javax.sql",
                    #"javax.xml",
                    #"javax.xml.datatype",
                    #"javax.xml.namespace",
                    #"javax.xml.parsers",
                    #"javax.xml.transform",
                    #"javax.xml.transform.dom",
                    #"javax.xml.transform.sax",
                    #"javax.xml.transform.stream",
                    #"javax.xml.validation",
                    #"javax.xml.xpath"
                    ]

        for pack in packages:
            populate(rootpath+javapath,pack,jpubclass)
            populate(rootpath+apath,pack,apubclass)
            #populate(gnupath,pack,gpubclass)

        allinter = jpubclass & apubclass
        print "js = %d, as = %d, gs = %d, inter = %d\n" % (len(jpubclass),len(apubclass),len(gpu
bclass),len(allinter))

        #return

        for pack in packages:
            print "java"
            topcount(rootpath+javapath,pack,jcdict,jperclass,jset,jfunclist,jprivdic
t,jmethnames,jpubclass)
            print "android"
            topcount(rootpath+apath,pack,acdict,aperclass,aset,afunclist,aprivdict,a
methnames,apubclass)
            #print "gnu"
            #topcount(gnupath,pack,gcdict,gperclass,gset,gfunclist,gprivdict,gmethna
mes,gpubclass)

        print "%d packages analyzed" % (len(packages))
        print "\nJava Analysis"
        juset = report(jcdict,jperclass,jfunclist,jprivdict,jmethnames)
        print "\nAndroid Analysis"
        auset = report(acdict,aperclass,afunclist,aprivdict,amethnames)
        print "\nGnuClasspath Analysis"
        report(gcdict,gperclass,gfunclist,gprivdict,gmethnames)
        print "\n-----"

        jmset = juset
        amset = auset
        inter = jmset&amset
        aonly = amset-jmset
        jonly = jmset-amset
```

```
        print "android only count = ",len(aonly),len(amset)
        print "java only count = ",len(jonly),len(jmset)
        print "android only"
        for i,n in enumerate(sorted(aonly)):
            print i,n
        print "java only"
        for i,n in enumerate(sorted(jonly)):
            print i,n

        # public classes that are different?
        japub = jpubclass - apubclass
        ajpub = apubclass - jpubclass
        print "javapub = %d, android pub = %d, j-a = %d, a-j = %d\n" % (len(jpubclass),len(apubcla
    ss), len(japub),len(ajpub))
        print "java public not in android"
        for nm in sorted(japub):
            print nm
        print "-----\n"
        print "android public not in java"
        for nm in sorted(ajpub):
            print nm
        print "-----\n"

        aaset = set()
        jjset = set()
        for cl in aprivdict:
            for meth in aprivdict[cl]:
                aaset.add(cl+"."+get_method(meth))

        for cl in jprivdict:
            for meth in jprivdict[cl]:
                jjset.add(cl+"."+get_method(meth))



        inter = aaset & jjset

        ro = 0
        wo = 0
        for nm in inter:
            print nm
            if nm.endswith("readObject"):
                ro += 1
            if nm.endswith("writeObject"):
                wo += 1
        print "size of inter ",len(inter)
        print "size of android ",len(aaset)
        print "size of java ",len(jjset)

        print "read/write object count", ro+wo

        privlog = open("privatemethodlog","w")

        privlog.close()




    #    print "common package/private"
    #    inter = jset&aset
    #    for name in inter:
    #        print name
    #
    #    print "\nAndroid\n------"
    #    for name in aset:
    #        print name
    #    print "\nJava\n------"
    #    for name in jset:
    #        print name
```



```


    if __name__ == "__main__":
        analyze()
```

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections,re

acdict = collections.defaultdict(int)
aperclass = collections.defaultdict(int)
aprivdict = {}
aset = set()
amethnames = []
apubclass = set()

jcdict = collections.defaultdict(int)
jperclass = collections.defaultdict(int)
jprivdict = {}
jset = set()
jmethnames = []
jpubclass = set()

gcdict = collections.defaultdict(int)
gperclass = collections.defaultdict(int)
gprivdict = {}
gset = set()
gmethnames = []
gpubclass = set()

afunclist = []
jfunclist = []
gfunclist = []

methnames = []

public_ids = ["public class",
              "public abstract class",
              "public interface",
              "protected class",
              "protected",
              "public"]

def is_func(line):
    if "new " in line:
        return False
    parts = line.split()
    if line.startswith("public") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    if line.startswith("private") and line.find(")") >= 0 and line.find("(") >= 0:
        return True
    return False


def getClass(path):
    '''
    path ends with .java, return class name preceding .java including preceding .
    e.g., for java/lang/Arrays, return .Arrays
    '''
    nm = path[:-5]
    index = nm.rfind("/")
    return "."+nm[index+1:]

def pubtrack(fname,pubclass,clname):
    f = open(fname)
    allText = f.read()
    changedText = re.sub(r"\s+"," ",allText)
    contents = changedText.split()
    for i in range(len(contents)-2):
        if contents[i] == "public" and contents[i+1] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "interface":
```



```
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "abstract" and contents[i+2] == "class":
            pubclass.add(clname)
            break
        if contents[i] == "public" and contents[i+1] == "final" and contents[i+2] == "class":
            pubclass.add(clname)
            break

def do_one(packname,onepath,cdict,perclass,cset,funclist,privdict,methnames,pubclass):

    if not onepath.endswith(".java"):
        return True
    if onepath.endswith("package-info.java"):
        return True


    class_name = getClass(onepath)
    #comment out line below 12/26, no difference
    #pubtrack(onepath,pubclass,packname+class_name)
    fullname = packname+class_name
    if not fullname in pubclass:
        print "rejected",fullname
        return True

    f = open(onepath)

    pcount = 0
    first = True
    public = False
    pubf = 0
    privf = 0
    for line in f:

        line = line.strip()

        if is_func(line):
            methnames.append(line)
            if line.startswith("public"):
                pubf += 1
            else:
                privf += 1
                nm = packname+class_name
                if not nm in privdict:
                    privdict[nm] = []
                privdict[nm].append(line)

        if first and line.startswith("class "):
            #print "class",onepath,line
            base = os.path.basename(onepath)
            cset.add(base)

        pfound = False
        for pub in public_ids:

            if line.startswith(pub):
                if first:
                    first = False
                    if line.find("public") >= 0 or line.find("protected") >= 0:
                        public = True
                    else:
                        print "big problem",onepath,pub,line
                if line.find("protected") < 0:
                    pcount += 1

                cdict[pub] += 1
                pfound = True
                if line.find("class") >= 0 and line.find("extends") >= 0:
                    cdict["extends"] += 1
                elif line.find("interface") >= 0 and line.find("extends") >= 0:
```

```
                    cdict["extends"] += 1
                break

        f.close()

        perclass[pcount] += 1
        if pcount == 0:
            #print "%s = %d" % (onepath,pcount)
            pass

        funclist.append((pubf,privf))
        return public


    def pop_one(packname,onepath,pubclass):

        if not onepath.endswith(".java"):
            return True
        if onepath.endswith("package-info.java"):
            return True

        class_name = getClass(onepath)
        pubtrack(onepath,pubclass,packname+class_name)

    def populate(basepath,packname,pubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = pop_one(packname,top_path,pubclass)


    def topcount(basepath,packname,cdict,perclass,cset,funclist,privdict,methnames,p
    ubclass):
        parts = packname.split(".")
        pathize = '/'.join(parts)
        packagepath = os.path.join(basepath,pathize)
        for top in os.listdir(packagepath):
            top_path = os.path.join(packagepath,top)
            if os.path.isdir(top_path):
                #print "*** %s is a directory in %s" % (top,packagepath)
                pass
            else:
                c = do_one(packname,top_path,cdict,perclass,cset,funclist,privdict,m
    ethnames,pubclass)
                if not c:
                    #print "no public",top_path,top
                    pass
                #print "%s has %d public" % (top_path,c)

    def func_stats(coll):
        low = 0
        word_total = 0
        wt_count = 0
        nonlow = 0
        getter = 0
        setter = 0
        req = 0
        equ = 0

        obj_names = ["toString", "hashCode", "notifyAll", "getClass", "equals","clone","wait","fin
    alize","notify"]

        for nm in coll:

            if nm in obj_names:
                req += 1
                if nm.startswith("eq"):
```



```
                    equ += 1
            elif nm.islower():
                low += 1
                #print "\t lower",nm
            else:
                wc = 0
                for i,ch in enumerate(nm):
                    if ch.isupper() and i > 0 and nm[i-1].islower():
                        wc += 1

                wc += 1
                #word_total += wc
                nonlow += 1

                if nm.startswith("get"):
                    getter += 1
                elif nm.startswith("set"):
                    setter += 1
                else:
                    word_total += wc
                    wt_count += 1

        ftot = low+nonlow+req
        print "total = %d, req = %d one = %d more = %d\n" % (ftot,req,low,nonlow)
        print "perc = %f avg = %f\n" % (1.0*low/(low+nonlow+1),1.0*word_total/(wt_count+1
    ))
        print "non simple = %d\n" % (wt_count)

        print "getter = %d, setter = %d, req = %d, equal = %d, total = %d\n" % (getter,setter,req,equ,r
    eq+getter+setter)

    def funcalyze(methnames):
        all_names = set()
        names = []
        for meth in methnames:
            if meth.startswith("public"):
                nameEnd = meth.find("(")
                if nameEnd == -1:
                    print "error on ",meth
                else:
                    name = meth[:nameEnd]
                    space = name.rfind(" ")
                    mname = name[space+1:]
                    all_names.add(mname)
                    names.append(mname)

        print "total = %d, unique = %d\n" % (len(names), len(all_names))
        print "unique"
        func_stats(all_names)
        print "total"
        func_stats(names)

        meth_counts = [(names.count(nm),nm) for nm in all_names]
        smc = sorted(meth_counts, reverse=True)
        print "top func occurrences"
        for pair in smc[:20]:
            print pair


        return all_names



    def report(cdict,perclass,funclist,privdict,methnames):

        uset = funcalyze(methnames)


        ctotal = 0
        for key in cdict:
            if key.find("public") < 0:
                continue
            print "%s occurrences = %d" % (key,cdict[key])
```

```
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "public class/interface total = %d" % (ctotal)

        ctotal = 0
        for key in cdict:
            if key.find("protected") < 0:
                continue
            print "%s occurrences = %d" % (key,cdict[key])
            if key.find("class") >= 0 or key.find("interface") >= 0:
                ctotal += cdict[key]
        print "----"
        print "protected class/interface total = %d" % (ctotal)

        print "per class method counts"
        print "# methods\t#classes"
        total = 0
        levels = collections.defaultdict(int)
        levlist = [0,1,6,11,16,21,51,101,100001]
        for method_count in sorted(perclass.keys()):
            print "%d\t%d" % (method_count,perclass[method_count])
            total += method_count*perclass[method_count]
            for lev in xrange(1,len(levlist)):
                if levlist[lev-1] <= method_count < levlist[lev]:
                    levels[lev] += perclass[method_count]
        print "-----"
        print "total methods = %d" % (total)
        print "\n---summary---"
        total = 0
        for lev in xrange(1,len(levlist)):
            print "perclass from %d to %d = %d" % (levlist[lev-1],levlist[lev]-1,levels[le
v])
            total += levels[lev]
        print "total = %d" % (total)

        print "size of funclist = %d" % (len(funclist))
        total = 0
        totalMeths = 0
        totalPriv = 0
        total2 = 0
        for x in funclist:
            totalMeths += x[0] + x[1]
            totalPriv += x[1]
            if x[0] != 0 or x[1] != 0:
                total += 100.0*x[0]/(x[1]+x[0])
                total2 += 100.0*x[1]/(x[1]+x[0])
        print "average = %f" % (total/(len(funclist)+1))
        print "total meths = %d" % (totalMeths)
        print "total private = %d" % (totalPriv)
        print "averagePRIV = %f" % (total2/(len(funclist)+1))
        return uset

    def analyze():
        rootpath = "/Volumes/My Passport for Mac/expert/google"
        #apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
        apath = "/Users/ola/expert/google/newsource/libcore/luni/src/main/java/"
        #apath = "/Users/ola/expert/google/S21"
        #javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
        javapath = "/Users/ola/expert/google/JDK/jdk/src/share/classes"
        gnupath = "/Users/ola/expert/google/source-gnu/classpath-0.98"

        packages = ["java.awt.font",
                    "java.beans",
                    "java.io",
                    "java.lang",
                    "java.lang.annotation",
                    "java.lang.ref",
                    "java.lang.reflect",
                    #"java.math",
                    "java.net",
                    "java.nio",
                    "java.nio.channels",
```



```
                    "java.nio.channels.spi",
                    "java.nio.charset",
                    "java.nio.charset.spi",
                    "java.security",
                    "java.security.acl",
                    "java.security.cert",
                    "java.security.interfaces",
                    "java.security.spec",
                    "java.sql",
                    "java.text",
                    "java.util",
                    #"java.util.concurrent",
                    #"java.util.concurrrent.atomic",
                    #"java.util.concurrent.locks",
                    "java.util.jar",
                    "java.util.logging",
                    "java.util.prefs",
                    "java.util.regex",
                    "java.util.zip",
                    "javax.crypto",
                    "javax.crypto.interfaces",
                    "javax.crypto.spec",
                    "javax.net",
                    "javax.net.ssl",
                    "javax.security.auth",
                    "javax.security.auth.callback",
                    "javax.security.auth.login",
                    "javax.security.auth.x500",
                    "javax.security.cert",
                    "javax.sql",
                    #"javax.xml",
                    #"javax.xml.datatype",
                    #"javax.xml.namespace",
                    #"javax.xml.parsers",
                    #"javax.xml.transform",
                    #"javax.xml.transform.dom",
                    #"javax.xml.transform.sax",
                    #"javax.xml.transform.stream",
                    #"javax.xml.validation",
                    #"javax.xml.xpath"
                    ]

        for pack in packages:
            populate(rootpath+javapath,pack,jpubclass)
            populate(rootpath+apath,pack,apubclass)
            #populate(gnupath,pack,gpubclass)

        allinter = jpubclass & apubclass
        print "js = %d, as = %d, gs = %d, inter = %d\n" % (len(jpubclass),len(apubclass),len(gpu
bclass),len(allinter))

        #return

        for pack in packages:
            print "java"
            topcount(rootpath+javapath,pack,jcdict,jperclass,jset,jfunclist,jprivdic
t,jmethnames,jpubclass)
            print "android"
            topcount(rootpath+apath,pack,acdict,aperclass,aset,afunclist,aprivdict,a
methnames,apubclass)
            #print "gnu"
            #topcount(gnupath,pack,gcdict,gperclass,gset,gfunclist,gprivdict,gmethna
mes,gpubclass)

        print "%d packages analyzed" % (len(packages))
        print "\nJava Analysis"
        juset = report(jcdict,jperclass,jfunclist,jprivdict,jmethnames)
        print "\nAndroid Analysis"
        auset = report(acdict,aperclass,afunclist,aprivdict,amethnames)
        print "\nGnuClasspath Analysis"
        report(gcdict,gperclass,gfunclist,gprivdict,gmethnames)
        print "\n-----"
```

```
        jmset = juset
        amset = auset
        inter = jmset&amset
        aonly = amset-jmset
        jonly = jmset-amset
        print "android only count = ",len(aonly),len(amset)
        print "java only count = ",len(jonly),len(jmset)
        print "android only"
        for i,n in enumerate(sorted(aonly)):
            print i,n
        print "java only"
        for i,n in enumerate(sorted(jonly)):
            print i,n

        # public classes that are different?
    #    japub = jpubclass - apubclass
    #    ajpub = apubclass - jpubclass
    #    print "javapub = %d, android pub = %d, j-a = %d, a-j = %d\n" % (len(jpubcla
    ss),len(apubclass), len(japub),len(ajpub))
    #    print "java public not in android"
    #    for nm in sorted(japub):
    #        print nm
    #    print "-----\n"
    #    print "android public not in java"
    #    for nm in sorted(ajpub):
    #        print nm
    #    print "-----\n"


        privlog = open("privatelog","w")
        for pack in aprivdict:
            if pack in jprivdict:
                line = "package class private {0!s}\n".format(pack)
                print "package class private %s" % (pack)
                privlog.write(line)
                for priv in aprivdict[pack]:
                    line = "\tAndroid {0!s}\n".format(priv)
                    privlog.write(line)
                    #print "\tAndroid %s" % (priv)
                    if priv in jprivdict[pack]:
                        privlog.write("\t\talso in Java\n")
                        #print "\t\talso in Java"
                for priv in jprivdict[pack]:
                    if not priv in aprivdict[pack]:
                        privlog.write("\tJava "+priv+"\n")
                        #print "\tJava %s" % (priv)
        privlog.close()




    #    print "common package/private"
    #    inter = jset&aset
    #    for name in inter:
    #        print name
    #
    #    print "\nAndroid\n------"
    #    for name in aset:
    #        print name
    #    print "\nJava\n------"
    #    for name in jset:
    #        print name



    if __name__ == "__main__":
        analyze()
```

```
'''
Created as part of work on expert report
for Google/Oracle for GreenbergTraurig

@author: ola
@copyright: owen astrachan, compsciconsulting
'''
import os,collections


def do_one(packname,onepath):
    '''
    return number of lines in onepath if a .java file
    '''

    if not onepath.endswith(".java"):
        return 0
    if onepath.endswith("package-info.java"):
        return 0

    f = open(onepath)

    lcount = 0
    for line in f:
        lcount += 1

    f.close()
    return lcount

def topcount(basepath,packname):
    parts = packname.split(".")
    pathize = '/'.join(parts)
    packagepath = os.path.join(basepath,pathize)
    total = 0
    ftot = 0
    for top in os.listdir(packagepath):
        top_path = os.path.join(packagepath,top)
        if os.path.isdir(top_path):
            #print "*** %s is a directory in %s" % (top,packagepath)
            pass
        else:
            c = do_one(packname,top_path)
            if c != 0:
                ftot += 1
                total += c
    return (total,ftot)

def analyze48():
    rootpath = "/Volumes/My Passport for Mac/expert/google"
    apath = "/Users/ola/expert/google/SOURCE/libcore/luni/src/main/java"
    javapath = "/Users/ola/expert/google/ESOURCE/j2se/src/share/classes"
    #gnupath = "/Users/ola/expert/google/source-gnu/classpath-0.98"
    javapath = "/Users/ola/expert/google/all/jdk15"
    #apath = "/Users/ola/expert/google/all/android"
    gnupath = "/Users/ola/expert/google/all/gnu"

    packages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                #"java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
```



```
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                #"java.util.concurrent",
                #"java.util.concurrrent.atomic",
                #"java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
                "javax.security.auth.callback",
                "javax.security.auth.login",
                "javax.security.auth.x500",
                "javax.security.cert",
                "javax.sql",
                #"javax.xml",
                #"javax.xml.datatype",
                #"javax.xml.namespace",
                #"javax.xml.parsers",
                #"javax.xml.transform",
                #"javax.xml.transform.dom",
                #"javax.xml.transform.sax",
                #"javax.xml.transform.stream",
                #"javax.xml.validation",
                #"javax.xml.xpath"
                ]

    allpackages = ["java.awt.font",
                "java.beans",
                "java.io",
                "java.lang",
                "java.lang.annotation",
                "java.lang.ref",
                "java.lang.reflect",
                "java.math",
                "java.net",
                "java.nio",
                "java.nio.channels",
                "java.nio.channels.spi",
                "java.nio.charset",
                "java.nio.charset.spi",
                "java.security",
                "java.security.acl",
                "java.security.cert",
                "java.security.interfaces",
                "java.security.spec",
                "java.sql",
                "java.text",
                "java.util",
                "java.util.concurrent",
                "java.util.concurrent.atomic",
                "java.util.concurrent.locks",
                "java.util.jar",
                "java.util.logging",
                "java.util.prefs",
                "java.util.regex",
                "java.util.zip",
                "javax.crypto",
                "javax.crypto.interfaces",
                "javax.crypto.spec",
                "javax.net",
                "javax.net.ssl",
                "javax.security.auth",
```

```
                    "javax.security.auth.callback",
                    "javax.security.auth.login",
                    "javax.security.auth.x500",
                    "javax.security.cert",
                    "javax.sql",
                    "javax.xml",
                    "javax.xml.datatype",
                    "javax.xml.namespace",
                    "javax.xml.parsers",
                    "javax.xml.transform",
                    "javax.xml.transform.dom",
                    "javax.xml.transform.sax",
                    "javax.xml.transform.stream",
                    "javax.xml.validation",
                    "javax.xml.xpath"
                    ]

    jpair = [0,0]
    apair = [0,0]
    gpair = [0,0]
    for pack in packages: #allpackages
        #j = topcount(rootpath+javapath,pack) #(javapath,pack)
        a = topcount(rootpath+apath,pack)
        #g = topcount(gnupath,pack)
        g = [0,0]
        j = [0,0]
        jpair[0] += j[0]
        jpair[1] += j[1]
        apair[0] += a[0]
        apair[1] += a[1]
        gpair[0] += g[0]
        gpair[1] += g[1]


    print "total # packages = %d\n" % (len(packages))
    print "Java = %d %d \nAndroid = %d %d\nGnu = %d %d\n" % (jpair[0],jpair[1],apair[0],a
pair[1],gpair[0],gpair[1])


if __name__ == "__main__":
    analyze48()
```

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE, INC. Defendant.	Case No. 3:10-cv-03561-WHA

DR. OWEN ASTRACHAN – OPENING REPORT ERRATA

I would like to correct the following typographical errors in my Opening Expert Report Of Dr. Owen Astrachan On Technical Issues Relating To Fair Use (January 8, 2016):

- ¶ 107: delete "java.math" from list
- ¶ 183: delete reference to "java.math" and subsequent explanatory paragraph
- ¶ 186: replace "a package called 'addingsubstractingandothernerdystuff' instead of the far more garden-variety choice 'java.math'" with "a class called 'java.lang.Addingsubstractingandothernerdystuff' instead of the far more garden-variety choice 'java.lang.Math'"
- ¶¶ 222, 223: replace "162" with "166"
- ¶ 274: add a period at the end of the paragraph
- ¶ 276: replace ":" with "."

Executed on the 13th of March, 2016 in San Francisco, CA.

Dr. Owen Astrachan